VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Popis prostředků pro zpracování obrazu v prostředí Scilab

Description of Tools for Image Processing in Scilab

2018 Bc. Kristýna Páleníková

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Zadání diplomové práce

Student: **Bc. Kristýna Páleníková**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2601T013 Telekomunikační technika

Téma: Popis prostředků pro zpracování obrazu v prostředí Scilab
Description of Tools for Image Processing in Scilab

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je provést rešerši nástrojů pro zpracování obrazu v opensourceovém prostředí Scilab a použití jednoho konkrétního nástroje (po dohodě s vedoucím práce) detailně popsat na příkladu kodéru formátu JPEG pro účely výuky.

1. Popište nástroje pro práci s obrazem v prostředí Scilab.
2. Popište detailně jednotlivé kroky komprese JPEG.
3. V prostředí Scilab vytvořte funkce pro realizaci jednotlivých kroků komprese JPEG. Jednotlivé dílčí funkce detailně zdokumentujte (např. formou komentářů uvnitř funkcí).
4. Otestujte funkčnost kodéru převodem obrázku z bezeztrátového formátu do vámi vytvořeného JPEG formátu a otevřením v nezávislém prohlížeči obrázků.

Seznam doporučené odborné literatury:

[1] UHLÍŘ, Jan; SOVKA, Pavel. *Číslicové zpracování signálů.* Vyd. 2. přeprac. Praha: Vydavatelství ČVUT, 2002. ISBN 80-01-02613-2.

[2] BRIGGS, William L. a Van Emden HENSON. *The DFT: an owner's manual for the discrete Fourier transform.* Philadelphia: Society for Industrial and Applied Mathematics, c1995. ISBN 0-89871-342-0.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Jan Skapa, Ph.D.**

Datum zadání: 01.09.2017

Datum odevzdání: 30.04.2018

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA * Fakulta elektrotechniky a informatiky * 1D16 * OSTRAVA *

doc. Ing. Miroslav Vozňák, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracovala samostatně. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

V Ostravě 30. dubna 2018 Pálníková .....

Ráda bych na tomto místě poděkovala všem, kteří mi s prací pomohli, hlavně Ing. Janu Skapovi, Ph.D. za cenné rady a připomínky při vypracování mé diplomové práce.

Abstrakt

Cílem této diplomové práce je provést rešerši nástrojů pro zpracování obrazu v opensourceovém prostředí Scilab, následně popsat jednotlivé kroky komprese. Pomocí jednoho konkrétního nástroje vytvořit kodér formátu JPEG pro účely výuky. Jednotlivé kroky komprese jsou dále detailně zpracovány a vysvětleny v podobě kódů s podrobným komentářem. Funkčnost byla testována převodem obrázku z bezztrátového formátu do vytvořeného JPEG formátu a otevřením v nezávislém prohlížeči obrázků.

Klíčová slova: Zpracování obrazu, Scilab, IPCV, JPEG, DCT

Abstract

This diploma thesis focuses on a research of Scilab open-source modules for image processing and exact procedures involved in image compression. Its practical implementation includes creation of a JPEG encoder for learning purposes. Each step involved is precisely explained and commented appropriately. Functionality of the prototype was tested by converting a lossless image format to JPEG and then analysing the output with a completely independent digital image analyzer.

Key Words: Image processing, Scilab, IPCV, JPEG, DCT

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

Úvod

Cílem této diplomové práce je provést rešerši nástrojů pro zpracování obrazu v opensourceovém prostředí Scilab, následně popsat jednotlivé kroky komprese JPEG. Pomocí jednoho konkrétního nástroje (po dohodě s vedoucím práce) vytvořit kodér formátu JPEG pro účely výuky.

První část práce byla věnována popisu dostupných nástrojů pro práci s obrazem v prostředí Scilab a byl vybrán pouze jeden nástroj. Výběr nástroje byl proveden na základě aktivity vývoje, dostupnosti dokumentace, četnosti aktualizací, počtu stažení a míry podpory s poslední verzí Scilabu. Tento nástroj byl otestován na dvou operačních systémech, a to MS Windows verze 10 a Linux (distribuce Ubuntu 16.04). Pro další práci byl použit operační systém MS Windows verze 10.

Druhá část práce byla věnována detailnímu popisu jednotlivých kroků komprese JPEG (převod z barevného prostoru RGB do YCbCr, dopředná diskrétní kosinová transformace, kvantizace DCT koeficientů, kódování, ukládání souboru do formátu JPEG).

Dále byly v prostředí Scilab vytvořeny funkce pro realizaci jednotlivých kroků komprese JPEG a následné uložení zpracovaných dat do formátu JPEG. Tyto funkce byly detailně okomentovány pomocí komentářů uvnitř funkce. Nakonec byla otestována funkčnost kodéru převodem obrázku z bezdrátového formátu do mnou vytvořeného JPEG formátu a následným otevřením v nezávislém prohlížeči obrázků.

1 Nástroje pro práci s obrazem v prostředí Scilab

V této diplomové práci se budeme věnovat Scilabu verze 6.0.0. Detailní instrukce pro instalaci softwaru naleznete v mé bakalářské práci Využití prostředků Scilab pro simulace zpracování signálů, kde je popsána instalace Scilab verze 5.5.2 na systému Windows 8 a na systému Linux – Ubuntu 14.04.

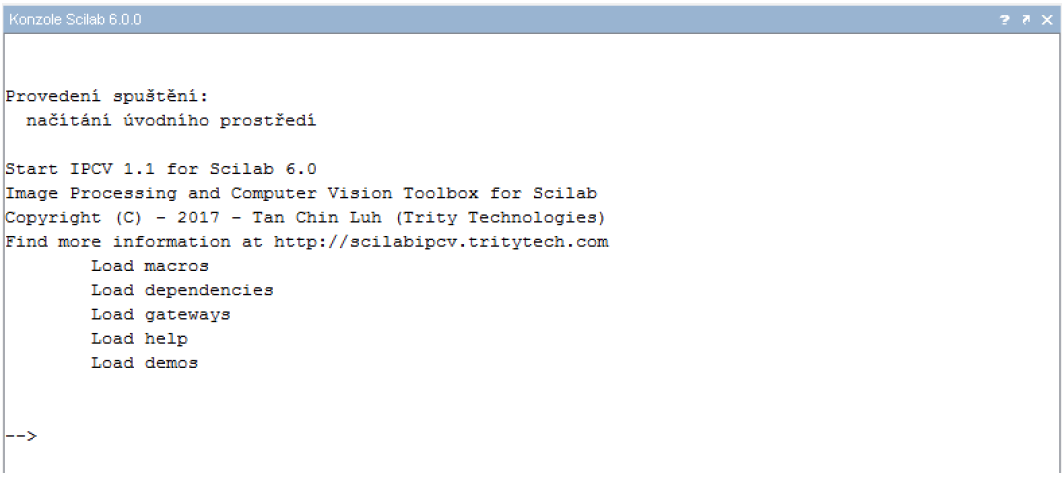
Nástroje pro práci s obrazem se nacházejí v *ATOMS* (AuTomatic mOdules Management for Scilab). Tento repozitář obsahuje rozšíření modulů Scilabu. V ATOMS nalezneme nejrůznější balíčky, příkladem může být balíček pro zpracování obrazu, grafické uživatelské rozhraní, zpracování signálu a jiné. Pro zpracování obrazu jsou dostupné tyto moduly:

- SIP – Scilab Image Processing Toolbox (2011),
- IPT2 – Image Processing Toolbox 2 (2013),
- SIVP – Scilab Image and Video Processing toolbox (2014),
- IWTT – Integer Wavelet Transform ToolBox (2015),
- OFIP – OpenCV Function Implementation through Python (2016),
- IPD – Image Processing Design Toolbox (2016),
- FCVT – Computer Vision Toolbox (2017),
- IPCV – Image Processing and Computer Vision Toolbox (2017),
- SCICV – Scilab Computer Vision Module (2017).

Při výběru nejvhodnějšího modulu byl kladen důraz na aktivitu vývoje, dostupnost dokumentace, četnost aktualizací, počet stažení a míru podpory s poslední verzí Scilabu. Všechna kritéria splňují moduly IPCV (2017) a SCICV (2017). Ostatní moduly nejsou podporovány v nejnovější verzi Scilabu. Při ověřování aktivity vývoje a dostupnosti dokumentace byl, po konzultaci s vedoucím práce, vybrán modul IPCV (2017). IPCV (2017) má oproti SCICV (2017) přehlednější dokumentaci, a také rozšířenou podporu, která může pomoci při řešení problémů.

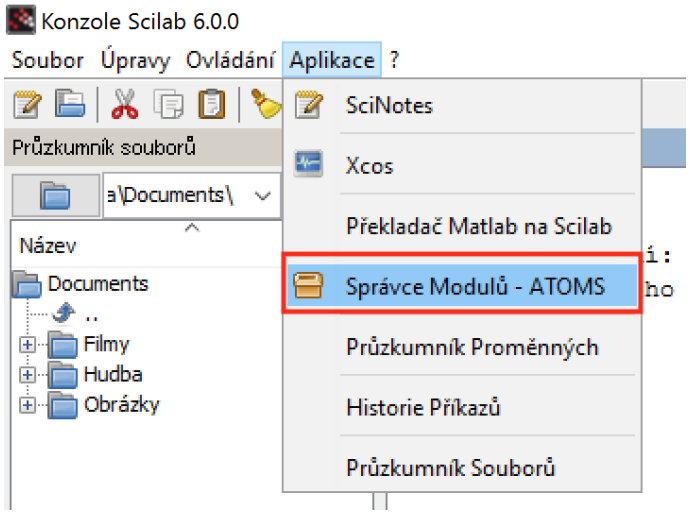
1.1 Instalace nástroje IPCV na systému Windows verze 10

Instalace nástroje IPCV na systému Windows může být provedena dvěma způsoby. Jedním ze způsobů je instalace příkazem `--> atomsInstall("IPCV")`, který napíšeme do konzole a balíček se nám nainstaluje. Poté program restartujeme. Po restartování se nám zobrazí načtený modul (obrázek 1).

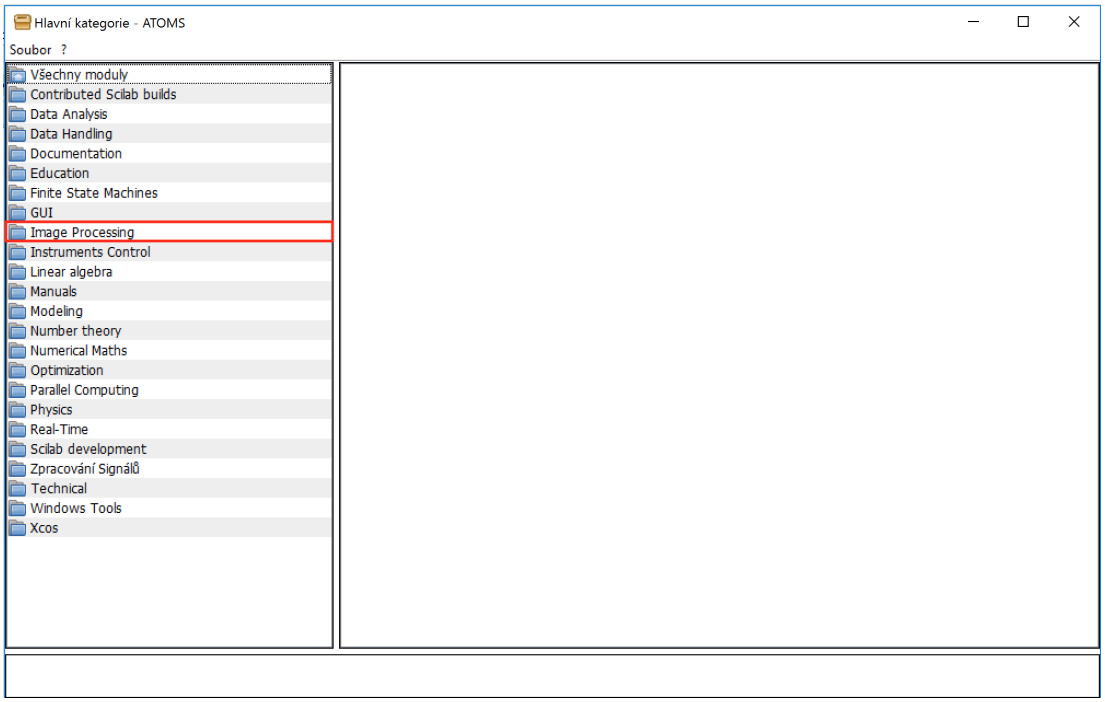


Obrázek 1: Načtení nástroje IPCV po instalaci na systému Windows.

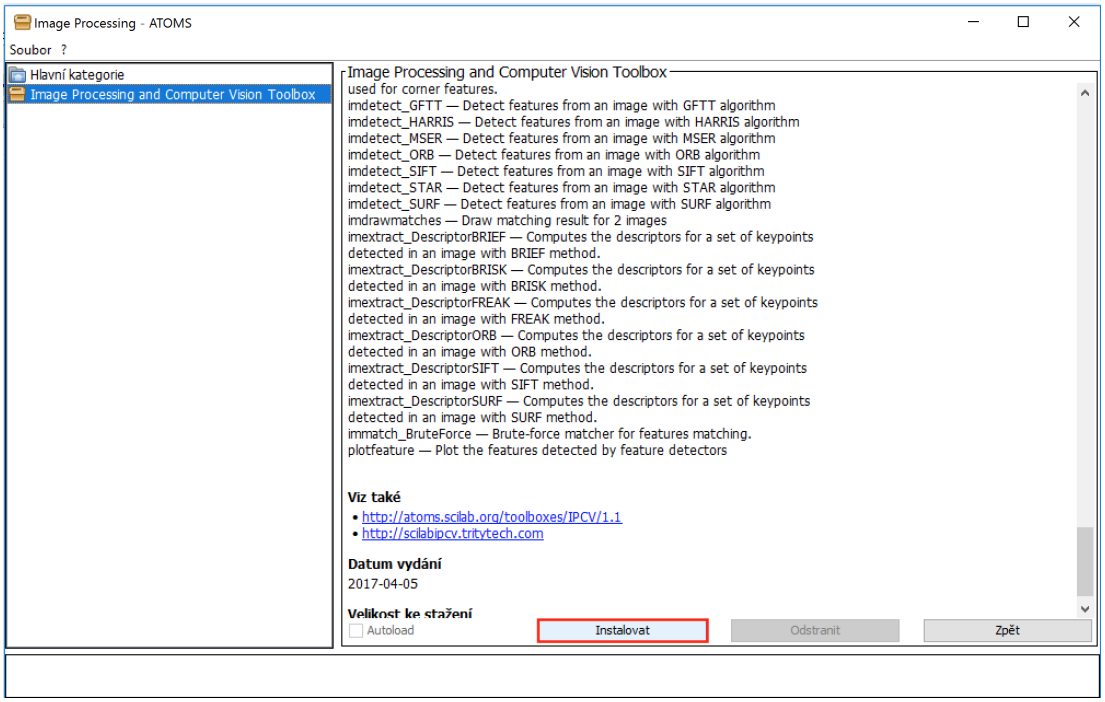
Druhý způsob je použití „klikátka“, což může být pro uživatele přívětivější. Najedeme myší na horní panel, klikneme na *Aplikace* a zvolíme *Správce Modulů - ATOMS* (obrázek 2). Otevře se okno, kde vidíme všechny dostupné moduly. Vybereme *Image Processing* (obrázek 3), a poté klikneme na *Image Processing and Computer Vision Toolbox* (obrázek 4). Dále klikneme na tlačítko *Instalovat*, a poté program restartujeme. Po restartování programu můžeme ve správci aplikací - ATOMS vidět nainstalované moduly (obrázek 5).



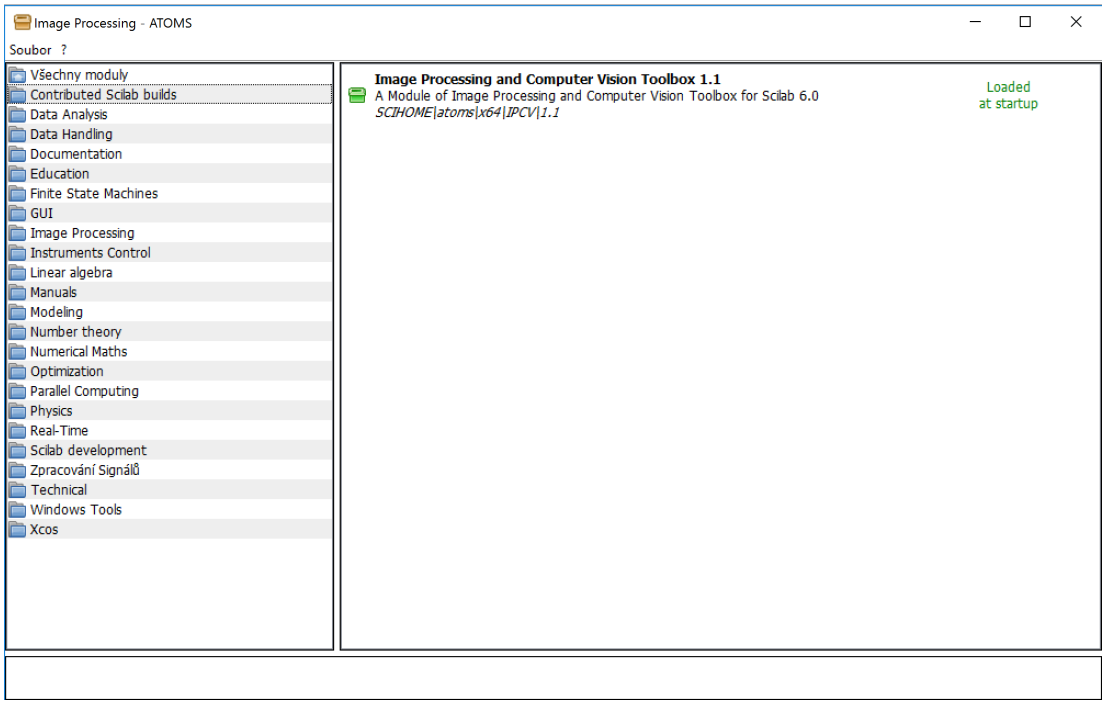
Obrázek 2: Instalace nástroje IPCV pomocí „klikátka“ – krok 1.



Obrázek 3: Instalace nástroje IPCV pomocí „klikátka“ – krok 2.



Obrázek 4: Instalace nástroje IPCV pomocí „klikátka“ – krok 3.



Obrázek 5: Načtené moduly ve Správci Modulů – ATOMS.

1.2 Instalace nástroje IPCV na systému Linux - Ubuntu 16.04

Instalace nástroje IPCV na systému Linux – Ubuntu 16.04 je trochu komplikovanější než na systému Windows. Prvním požadavkem je mít nainstalované vývojové soubory pro knihovnu OpenCV. Do okna terminálu napíšeme příkaz **`sudo -i`**, který nás přepne na root uživatele. Root uživatel má oprávnění vykonávat více operací, jako je například instalace programů. Nyní nainstalujeme vývojové soubory. Použijeme příkaz **`apt-get install libopencv-dev`**, dále přidáme nonfree repozitář pomocí příkazu **`add-apt-repository --yes ppa:xqms/opencv-nonfree`** (tento repozitář není součástí systému). Poté aktualizujeme systém příkazem **`apt-get update`** a nainstalujeme nonfree vývojový soubor příkazem **`apt-get install libopencv-nonfree-dev`**.

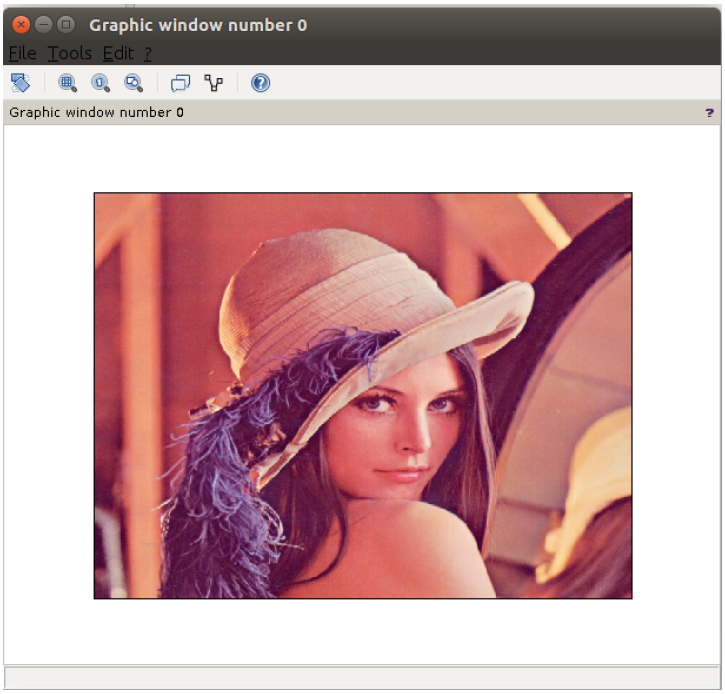
Druhým požadavkem je odstranění všech **`libgomp*`** ve složce **`scilab-6-0-0/lib/thirdparty/redist`**. Je to z důvodu správné funkčnosti modulu IPCV. Pokud dané soubory neodstraníme, tak se nám modul nenačte správně a nebudeme ho moci používat.

Po zvládnutí prvního i druhého požadavku spustíme program a nainstalujeme daný modul. Opět ho můžeme nainstalovat dvěma způsoby, jak bylo uvedeno v předchozí podkapitole 1.1.

Pro vyzkoušení funkcionality modulu IPCV si stáhneme na plochu obrázek, v mém případě se jedná o obrázek s názvem Lenna.png. Po spuštění programu provedeme následující příkazy:

```
-->s=imread("/home/parallels/Desktop/Lenna.png") // ulozeni obrazku do matice
-->imshow(s) // zobrazeni nacteneho obrazku
```

Výpis 1: Uložení a zobrazení obrázku



Obrázek 6: Načtení obrázku pomocí nástroje IPCV.

2 Matematický aparát

Dříve, než se budeme bavit o kompresi JPEG, kde se používá diskrétní kosinová transformace, je vhodné si připomenout rozvoj funkce ve Fourierovu řadu a diskrétní Fourierovu transformaci.

2.1 Fourierovy řady

Zopakujme si vztahy pro rozvoj signálu ve Fourierovu řadu [2],[15]. Každý periodický signál, který je energetický (má konečnou energii) na základní periodě T_0, lze rozvést do součtu harmonických funkcí s frekvencemi odpovídajícími celočíselným násobkům základní frekvence $f_0 = 1/T_0$. *Komplexní tvar Fourierovy řady* lze zapsat vztahem pro syntézu signálu:

$$x(t) = \sum_{k=-\infty}^{\infty} \hat{c}[f_0 k] e^{i2\pi f_0 k\ t}. \tag{1}$$

Koeficienty $\hat{c}[0]$ a $\hat{c}[f_0 k]$ pak spočteme jako

$$\hat{c}[0] = \frac{1}{T_0} \int_{T_0} x(t) dt, \tag{2}$$

$$\hat{c}[f_0 k] = \frac{1}{T_0} \int_{T_0} x(t) e^{-i2\pi f_0 k\ t} dt. \tag{3}$$

Koeficient $\hat{c}[0]$ nazýváme *stejnosměrná složka* signálu $x(t)$. Člen $1/T_0$ je ortonormalizační koeficient. Odtud lze odvodit *reálný tvar Fourierovu řady*

$$x(t) = \hat{a}[0] + \sum_{k=1}^{\infty} \hat{a}[f_0 k] \cos(2\pi f_0 k\ t) + \hat{b}[f_0 k] \sin(2\pi f_0 k\ t). \tag{4}$$

Koeficienty $\hat{a}[f_0 k]$ a $\hat{b}[f_0 k]$ spočteme následovně:

$$\hat{a}[0] = \frac{1}{2} \cdot \frac{2}{T_0} \int_{T_0} x(t) dt, \tag{5}$$

$$\hat{a}[f_0 k] = \frac{2}{T_0} \int_{T_0} x(t) \cos(2\pi f_0 k\ t) dt, \tag{6}$$

$$\hat{b}[f_0 k] = \frac{2}{T_0} \int_{T_0} x(t) \sin(2\pi f_0 k\ t) dt, \tag{7}$$

kde integrujeme přes dobu trvání základní periody T_0. Koeficient $\hat{a}[0]$ nazýváme *stejnosměrná složka* signálu $x(t)$.

Při rekonstrukci signálu $x(t)$ z koeficientů Fourierovy řady dochází k periodickému rozšíření signálu na celou množinu $\mathbb{R}$. Při výpočtu koeficientů můžeme záměrně signál rozšířit periodicky tak, že vznikne funkce sudá (sudé periodické prodloužení). Toto rozšíření je vhodné proto, že je spojité v krajních bodech původního intervalu. V tomto případě budou všechny koeficienty $\hat{b}[f_0 k]$ rovny nule. Vyruší se tedy všechny sinové složky a zůstanou pouze složky kosinové. Toho lze využít ke snížení množství potřebných koeficientů.

2.2 Diskrétní Fourierova transformace (1D DFT)

Diskrétní Fourierova transformace je diskrétní obdobou rozvoje signálu v součet Fourierovy řady [4],[16]. Uvažujme diskretizovaný vztah pro výpočet koeficientů Fourierovy řady. Pak platí, že vstupní signál je roven periodické diskrétní posloupnosti $x[n] = x(nT_{vz}) = \{x[0], x[1], \ldots, x[N-1]\}$ a $n = 0, 1, \ldots, N-1$,

$$x[n] = \sum_{k=0}^{N-1} \hat{c}[k] e^{i2\pi \frac{1}{N} kn}, \tag{8}$$

a pak diskrétní Fourierova transformace signálu $x[n]$ (koeficienty Fourierovy řady) je definována jako

$$\hat{c}[k] = \frac{1}{N} \sum_{n=0}^{N-1} x[n] e^{-i2\pi \frac{1}{N} kn}. \tag{9}$$

Člen $1/N$ před sumou je ortonormalizační koeficient. Koeficienty jsou rozloženy na frekvenční ose ve vzdálenosti $1/N$ od sebe navzájem, a tedy $1/N$ odpovídá základní frekvenci signálu. Vektor o složkách $|\hat{c}[k]|$ nazýváme magnitudovým spektrem a vektor o složkách $\arg(\hat{c}[k])$ fázovým spektrem signálu $x[n]$.

2.3 Diskrétní kosinová transformace (1D DCT)

Jedná se o speciální případ diskrétní Fourierovy transformace. Jde o obdobu Fourierovy řady v reálném tvaru pro sudé periodické prodloužení signálu [3].

Jednotlivé hodnoty vzorků signálu označme symbolem $x[n]$, kde n je pozice vzorku (index) v signálu. Po aplikaci jednorozměrné diskrétní kosinové transformace získáme sadu transformovaných koeficientů, které budeme značit $\hat{a}[k]$:

$$\hat{a}[k] = q[k] \sum_{n=0}^{N-1} x[n] \cos \frac{(2n+1)k\pi}{2N}, \tag{10}$$

kde N značí počet vzorků signálu $x[n]$ a hodnota pozice vzorku $k = 0, 1, \ldots, N-1$.

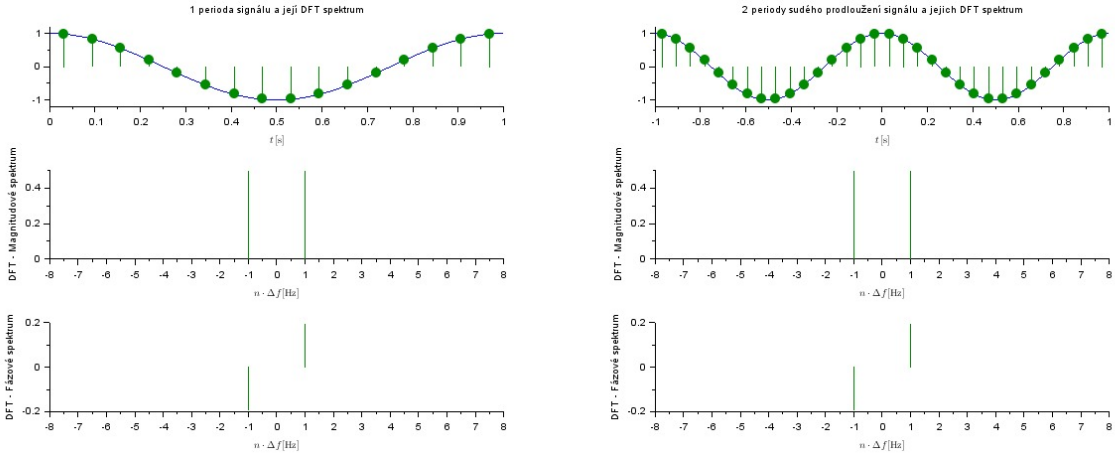
$$q[k] = \begin{cases} \sqrt{\frac{1}{N}} & \text{pro } k \ = 0, \\ \sqrt{\frac{2}{N}} & \text{jinak.} \end{cases}$$

Tento vztah můžeme zapsat také s využitím Kroneckerova δ, pro něž platí

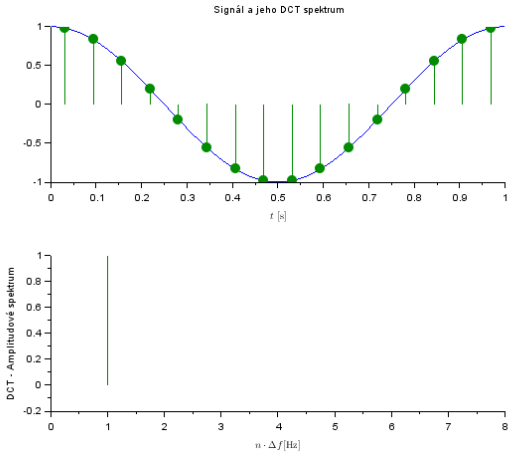
$$\delta_{nk} = \begin{cases} 1 & \text{pro } k \ = n, \\ 0 & \text{jinde}, \end{cases}$$

ve tvaru

$$\hat{a}[k] = \sqrt{\frac{1}{1+\delta_{0k}} \cdot \frac{2}{N}} \sum_{n=0}^{N-1} x[n] \cos \frac{(2n+1)k\pi}{2N}. \tag{11}$$



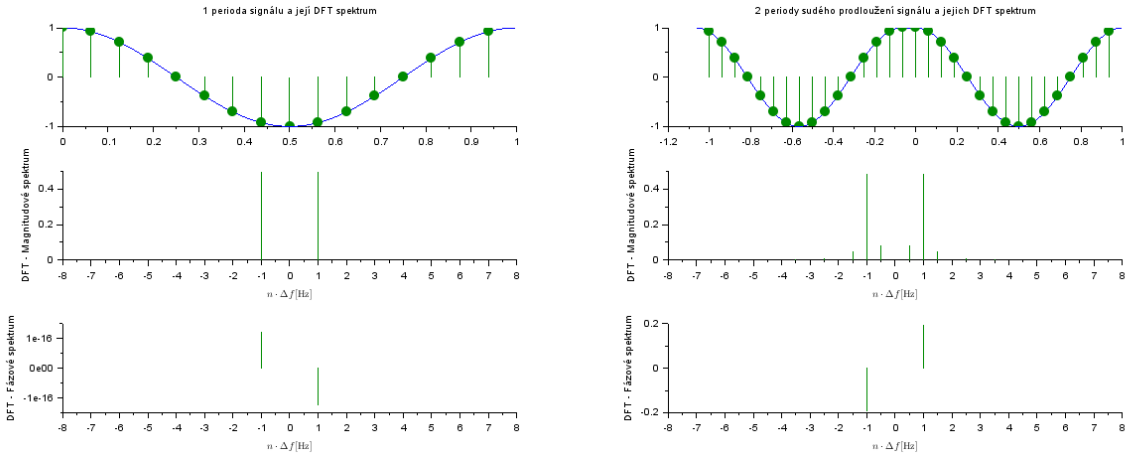
Obrázek 7: 1 perioda signálu (vlevo) a jeho sudé prodloužení (vpravo) a jejich DFT spektra.



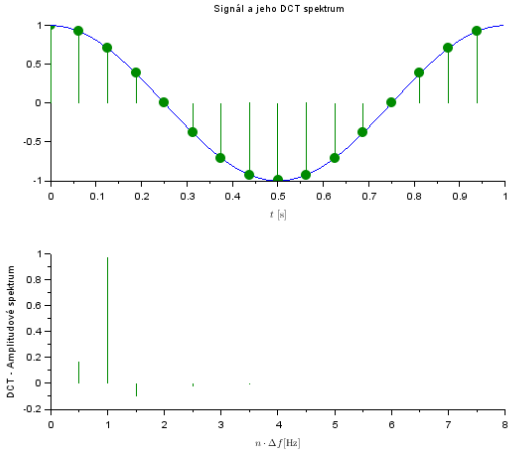
Obrázek 8: 1 perioda signálu a její DCT spektrum.

Na obrázku 7 vlevo můžeme vidět 1 periodu kosinové vlny a její DFT spektrum. Magnitudové spektrum ukazuje velikost koeficientů Fourierovy řady. Fázové spektrum zobrazuje argumenty koeficientů Fourierovy řady. Vpravo můžeme vidět sudé prodloužení signálu a jeho DFT spektrum. Zde můžeme vidět, že jsme zvolili správné krokování při odebírání vzorků.

Na obrázku 8 vidíme 1 periodu sudého signálu a její DCT spektrum. DCT spektrum obsahuje pouze amplitudové spektrum, protože amplitudy mohou nabývat pouze kladných nebo záporných hodnot. Fázové spektrum tedy není zapotřebí.



Obrázek 9: 1 perioda signálu (vlevo) a jeho sudé prodloužení (vpravo) a jejich DFT spektra (jiné vzorkovací časy).



Obrázek 10: 1 perioda signálu a její DCT spektrum (jiné vzorkovací časy).

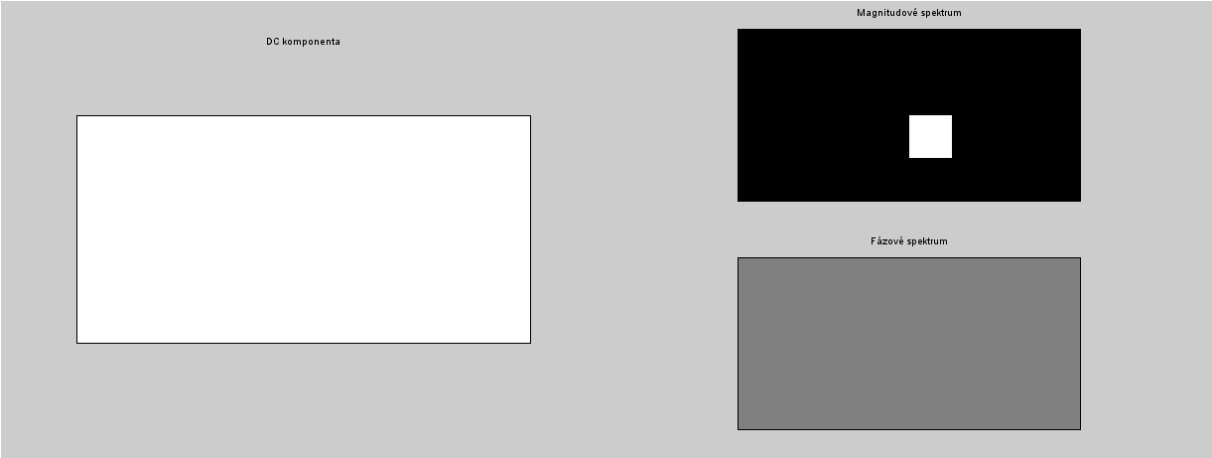
Na obrázku 9 vlevo můžeme vidět jiný krok vzorkování. Jde vidět, že pro jednu periodu signálu se DFT provede správně. Pokud se zaměříme na pravou část, kde máme sudé prodloužení signálu, vidíme v magnitudovém spektru postranní špičky. Tyto špičky se v původním spektru nevyskytovaly. Je to způsobené jevem Leakage (prosakování ve spektru). V případě DCT na obrázku 10, kde se počítá na sudém prodloužení, také dojde k leakage (první vzorek, který je na indexu 0, se při prodloužení zdvojí). Tento jev se nedá v reálných aplikacích odstranit, ale je možné ho eliminovat použitím okenních funkcí.

2.4 Dvourozměrná diskrétní Fourierova transformace (2D DFT)

Dvourozměrná diskrétní Fourierova transformace vychází z jednorozměrné DFT. Pokud tedy zobecníme rovnici 9, bude mít 2D DFT následující tvar:

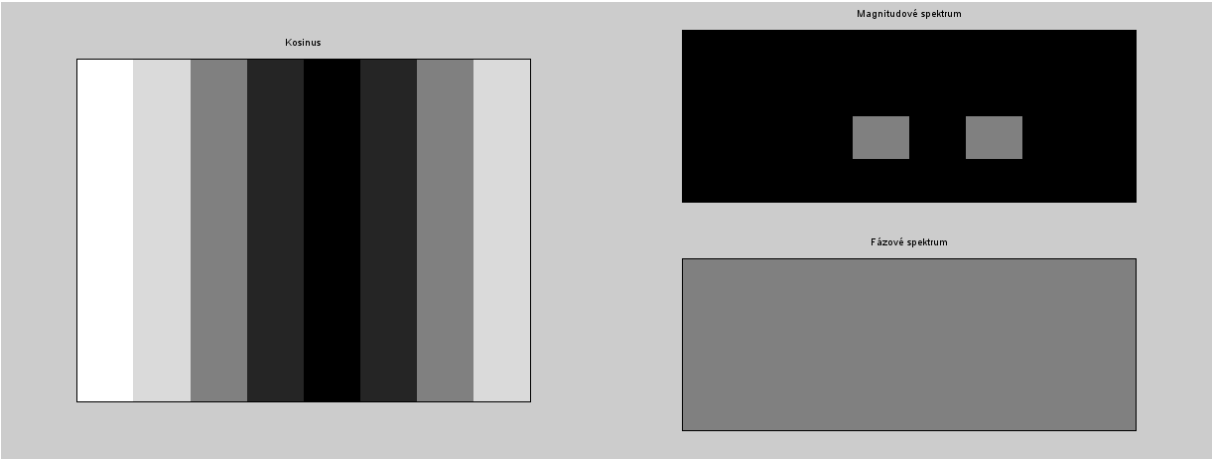
$$\hat{c}[u,v] = \frac{1}{M} \cdot \frac{1}{N} \cdot \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} x[m,n] e^{-i2\pi \frac{1}{M} \frac{1}{N} u \ v \ nm}. \tag{12}$$

Při výpočtu tedy procházíme nejprve řádky a pak sloupce matice dat (signálu) $x[m,n]$.



Obrázek 11: DC složka a její DFT spektrum.

Na obrázku 11 vlevo můžeme vidět DC komponentu a vpravo její DFT spektrum. Magnitudové spektrum obsahuje koeficient DC komponenty, který je umístěn na pozici nulové prostorové frekvence.

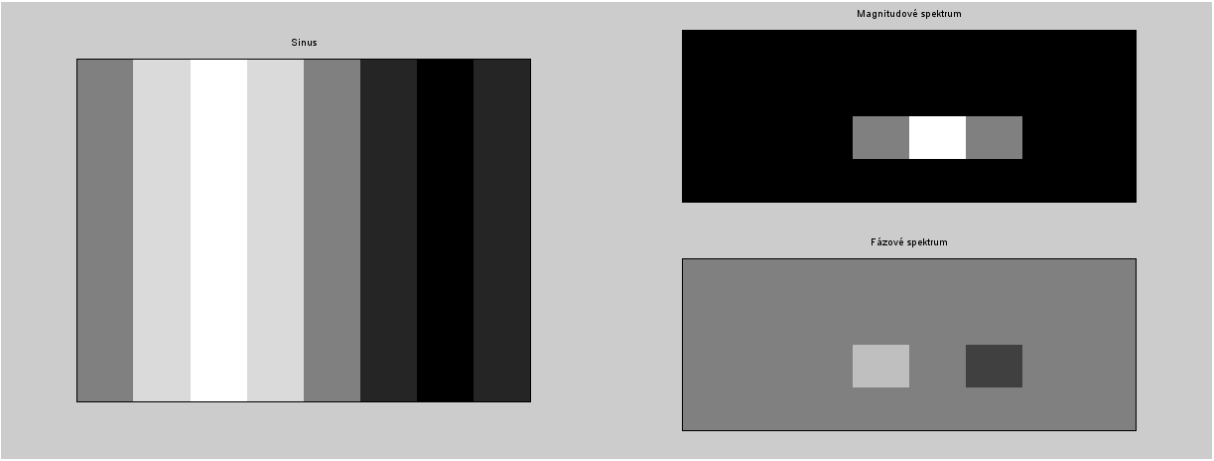


Obrázek 12: Kosinová vlna a její DFT spektrum.

```
f_sp = 1;// prostorova frekvence
tilt_deg = 0;// sklon
phase_shift = 0;// fazovy posun
z_1 = cos(2*%pi*f_sp*(x_1*cosd(tilt_deg) + y_1*sind(tilt_deg)) - phase_shift);
```

Výpis 2: Implementace kosinové vlny.

Pokud se podíváme na obrázek 12, který zobrazuje kosinovou vlnu (vlevo) a její DFT spektrum (vpravo), dle výše uvedené implementace můžeme vidět generování vlny. Na magnitudovém spektru si můžeme všimnout dvojice koeficientů odpovídajících harmonické složce.



Obrázek 13: Sinová vlna a její DFT spektrum.

```
f_sp = 1; // prostorova frekvence
tilt_deg = 0; // sklon
phase_shift = 0; // fazovy posuv
z_2 = 1 + sin(2*%pi*f_sp*(x_1*cosd(tilt_deg) + y_1*sind(tilt_deg)) - phase_shift);
```

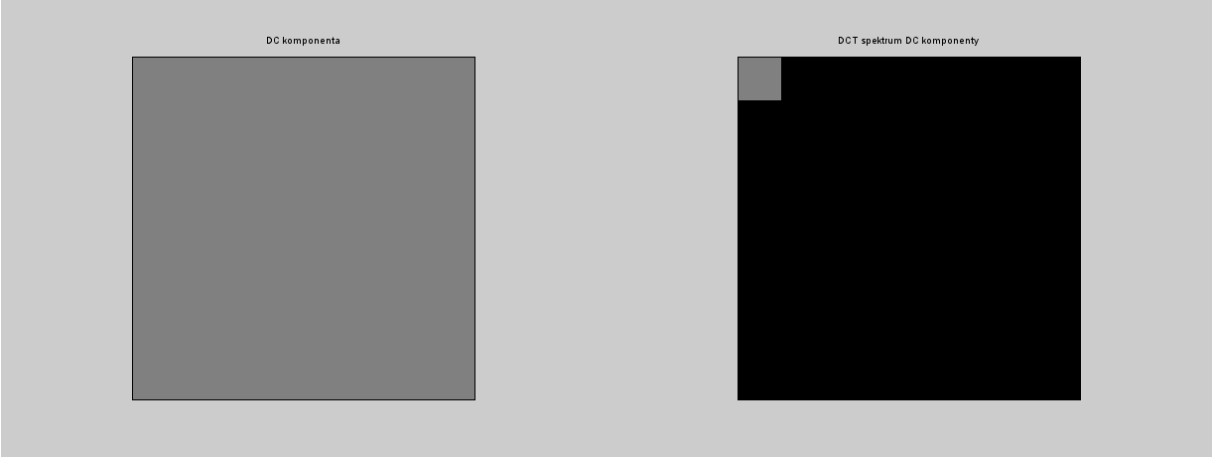
Výpis 3: Implementace sinové vlny.

Na obrázku 13 můžeme vidět sinovou vlnu (vlevo) a její DFT spektrum (vpravo). Magnitudové spektrum obsahuje koeficient stejnosměrné složky o velikosti jedna umístěné na pozici jednotkové prostorové frekvence a dále dvojice koeficientů odpovídajících harmonické složce. Fázové spektrum obsahuje dvojici bodů opačných hodnot fází odpovídajících dvojici komplexně sdružených koeficientů.

2.5 Dvourozměrná diskrétní kosinová transformace (2D DCT)

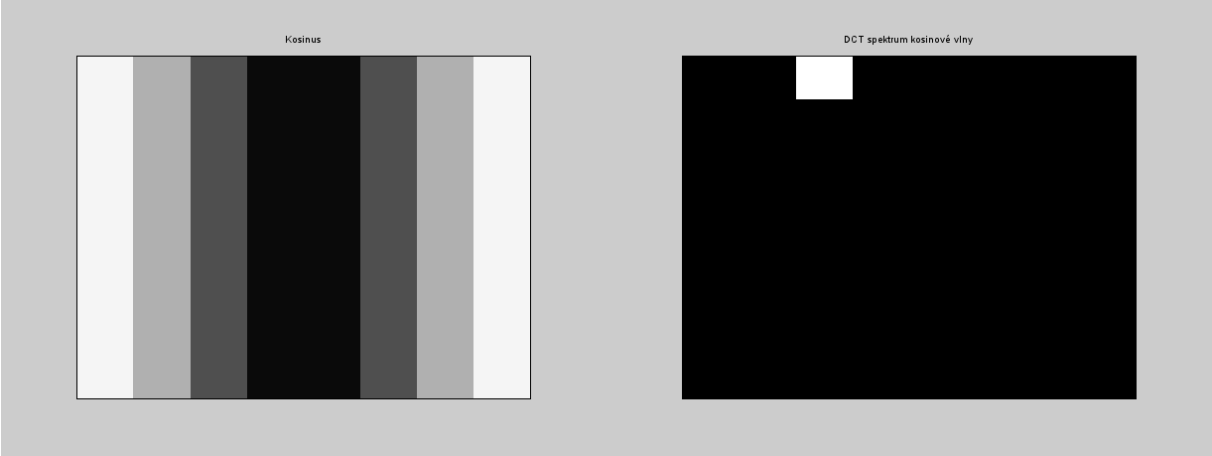
Dvourozměrná diskrétní kosinová transformace se vytváří zobecněním jednorozměrné DCT [11] [5]. Vstupem do 2D DCT je vzorkovaný signál uložený jako matice, který označme symbolem $x[m,n]$. Výstupem DCT je taktéž matice hodnot $\hat{a}[u,v]$.

$$\hat{a}[u,v] = \sqrt{\frac{1}{1+\delta_{0u}} \cdot \frac{2}{M}} \cdot \sqrt{\frac{1}{1+\delta_{0v}} \cdot \frac{2}{N}} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} x[m,n] \cos \frac{(2m+1)u\pi}{2M} \cos \frac{(2n+1)v\pi}{2N}. \quad (13)$$



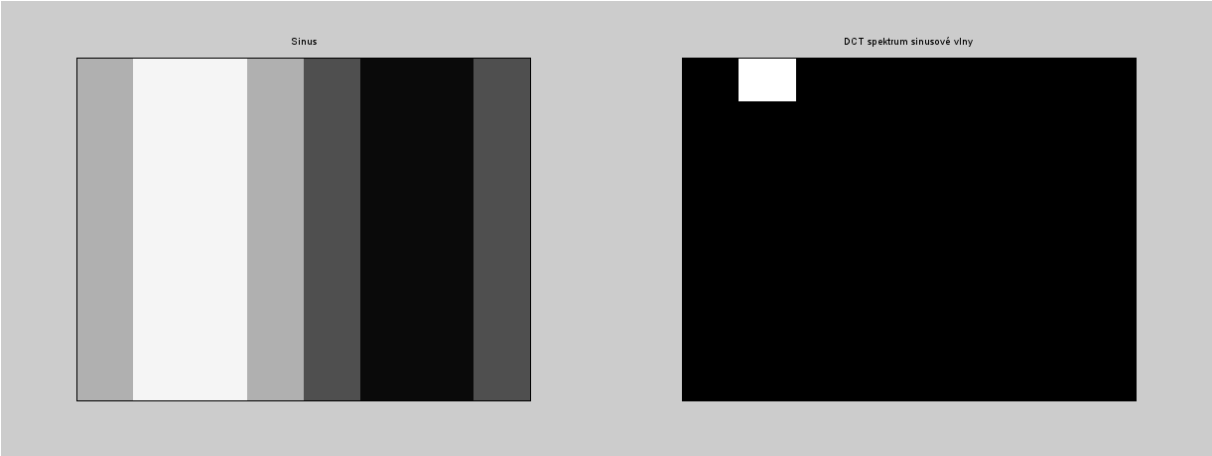
Obrázek 14: DC komponenta a její DCT spektrum.

Na obrázku 14 můžeme vidět DC komponentu (vlevo) a její DCT spektrum (vpravo). Stejnosměrná složka je na indexech (0,0). Dále následují nulové hodnoty, protože zde nemáme žádné střídavé složky.



Obrázek 15: Kosinová vlna a její DCT spektrum.

Na obrázku 15 je zobrazena kosinová vlna (vlevo) a její DCT spektrum (vpravo). Zde se vyskytuje právě jedna střídavá složka, která odpovídá vstupní kosinové vlně.



Obrázek 16: Sinová vlna a její DCT spektrum.

Na obrázku 16 se nachází sinová vlna (vlevo) a její DCT spektrum (vpravo). Zde si můžeme opět všimnout jen jedné střídavé složky, která odpovídá vstupní sinové vlně.

3 Komprese JPEG

Existuje velké množství grafických formátů pro ukládání digitální grafiky. Grafické formáty stanovují pravidla, podle kterých je obrázek uložen v souboru. Formáty se rozdělují na základě způsobu uložení grafické informace, dělí se na rastrové a vektorové formáty.

1. *Rastrové formáty* popisují matici barevných bodů, tedy bitmapu. Obraz se skládá z jednotlivých pixelů, kde každý pixel má definovanou barvu. Rastrové formáty dělíme na bezztrátové, ztrátové a ztrátově neutrální.

   (a) Bezztrátové:
   - BMP (Windows Bitmap),
   - DNG (Digital Negative),
   - PNG (Portable Network Graphics),
   - GIF (Graphics Interchange Format),
   - TIFF (Tagged Image File Format).

   (b) Ztrátové:
   - JPEG (Joint Photographics Experts Group).

   (c) Ztrátově neutrální:
   - EXIF (Exchangeable Image File),
   - WDP (Windows Media Photo).

2. *Vektorové formáty* popisují objekty v obraze pomocí příkazů a rovnic. Obraz se skládá z jednotlivých geometrických objektů, příkladem může být obdélník, elipsa, křivka apod. Každý objekt má definovanou barvu, styl obrysu a výplň:
   - SVG (Scalable Vector Graphics),
   - EPS (PostScript),
   - PDF (Portable Document Format).

JPEG jako formát obrázků

Jedním z nejrozšířenějších formátů souborů je JFIF [8](JPEG File Interchange Format), také známý jako JPEG (Joint Photographic Experts Group), což je konsorcium, které kompresi navrhlo. Základní kompresní schéma JPEG bylo schváleno v roce 1992, a následně, v roce 1994, došlo k definitivnímu schválení schématu pod označením ISO/IEC 10918–1. Nejběžnějšími příponami formátu JPEG jsou .jpg, .jpeg, .jfif, .jpe. Formát JPEG se používá především pro ukládání fotografických snímků nebo maleb realistických scenérií s hladkými přechody v tónu a barvě. Jsou to obrázky, které zobrazují reálný svět a vznikají většinou jako fotografie z digitálního fotoaparátu, pomocí mobilních telefonů, nebo jiným způsobem digitalizace obrazu.

Na druhou stranu je formát JPEG nevhodný pro obrázky s velkými jednobarevnými plochami a ostrými hranami. Není ani vhodný pro perokresbu, zobrazení textu nebo ikonky, protože kompresní metoda JPEG vytváří v takovém obrazu viditelné a rušivé artefakty. Na obrázku se objeví malé čtverečky. Formát JPEG byl spolu s formátem GIF běžně nejpoužívanějším formátem pro kompresi obrázků na webu, jako jsou designové prvky a loga. V dnešní době je vytlačován formátem PNG.

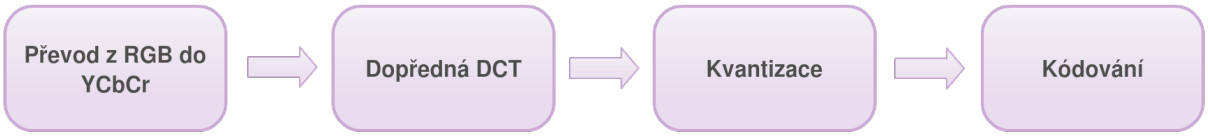
JPEG je ztrátová komprese, kde je cílem ukládat data úsporně a dosáhnout toho, aby rozdíl mezi uloženými a původními daty byl téměř nepostřehnutelný. Výsledek se sice nerovná původním datům, ale je jim velice podobný a s ohledem na nedokonalost lidských smyslů je rozdíl nevýznamný. Ztrátová komprese má nízkou výpočetní náročnost. Uplatňuje se především u obrazových a zvukových dat. Výhodou je možnost velkých kompresních poměrů za cenu minimální ztráty kvality. Požadovaná kvalita obrazu se pohybuje v rozmezí 7:1 až 30:1. Při ukládání obrázků pomocí JPEG komprese je možné nastavit tři parametry: mód komprese, kvalitu a způsob uložení barev.

Při nastavení módu komprese je nejčastěji využíván sekvenční, neboli základní mód. Dále je možné použít nastavení progresivního módu komprese. Tento mód ukládá obrázek tak, že je možno jej rekonstruovat od počátečního úseku souboru celý, ale v horší kvalitě. Kvalita se postupně zvyšuje s postupným načítáním obrázku. Tento mód se využívá v případě, že má uživatel pomalé připojení k internetu. Načtení obrázku netrvá dlouho, ale jeho kvalita je horší.

Dále je možné nastavit kvalitu a způsob uložení barev. Toto nastavení se projeví v kvalitě a velikosti souboru. Uživatel si zvolí, jak velkou ztrátu chce.

Jednotlivé kroky JPEG komprese

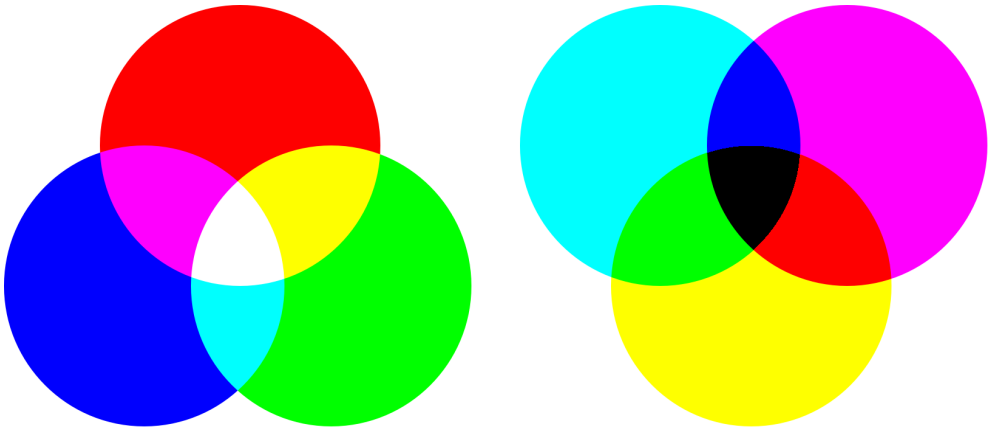
Komprese probíhá v několika krocích. Tyto kroky si v této kapitole detailně probereme. Na následujícím obrázku 17 můžeme vidět základní posloupnost kroků komprese. Po této posloupnosti kroků následuje ukládání samotného obrázku do formátu JPEG. Tento proces si také detailně popíšeme v kapitole 4. Jednotlivé implementace, které budou v této kapitole uvedeny, se budou týkat pouze jasové složky Y (kromě převodu z RGB do YCbCr).



Obrázek 17: Jednotlivé kroky JPEG komprese.

3.1 Převod RGB do YCbCr

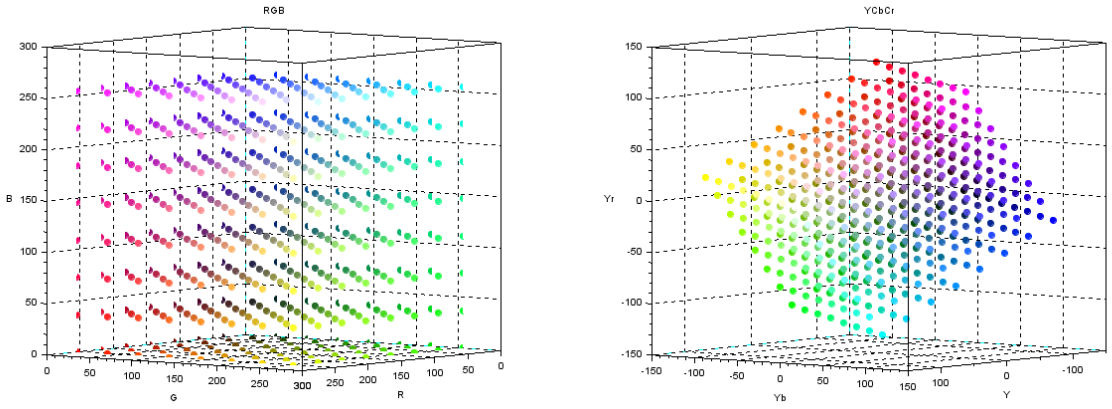
Prvním krokem JPEG komprese je převod z barevného prostoru RGB do YCbCr [1], [9]. U barevného modelu RGB (Red-Green-Blue) se bavíme o aditivním způsobu míchání barev, při němž se jednotlivé složky barev sčítají a vytvářejí světlo větší intenzity. RGB se používá v projektorech nebo v barevných monitorech. Nepotřebuje vnější světlo, tzn. monitor svítí i v naprosté tmě, na rozdíl od barevného modelu CMYK. Model CMYK je založený na subtraktivním míchání barev. Můžeme říci, že jde o míchání barev, kde se s každou další přidanou barvou ubírá část původního světla. Oba modely jsou zobrazeny na obrázku 18.



Obrázek 18: RGB model (vlevo) a CMYK model (vpravo).

Pokud se bavíme o RGB modelu, tak smícháním červené a modré barvy dostaneme purpurovou barvu. Smícháním modré a zelené barvy vznikne barva modrozelená. Spojením barvy zelené a červené získáme barvu žlutou. Smícháním všech tří základních barev dostaneme bílou barvu. V případě CMYK modelu spojením modrozelené a žluté barvy získáme barvu zelenou. Smícháním žluté a purpurové barvy dostaneme červenou barvu a smícháním purpurové a modrozelené vznikne barva modrá. Smícháním všech tří základních barev dostaneme barvu černou.

YCbCr je jeden z dalších barevných modelů, který můžeme najít i pod názvem YUV. Používá se zejména u videa a digitálních fotografií. Prostor YCbCr se skládá ze složek Y, Cb a Cr. Složka Y nese informaci o světlosti pixelů (jas) bez ohledu na jejich barvu. Složky Cb a Cr jsou barvonosné složky, kde Cb je modrá složka a Cr je červená složka. Lidské oko je více citlivé na změny jasu Y než na změny barvonosných složek Cb a Cr. Na obrázku 19 je vidět spojitost mezi barvovými prostory RGB a YCbCr. Převod jednotlivých barvových složek R, G a B na složky Y, Cb a Cr se provádí dle rovnic (14), (15) a (16).



Obrázek 19: Vykreslení RGB kostky v RGB prostoru (vlevo) a v YCbCr prostoru (vpravo) – výpis 20.

$$Y = 0,299 \cdot R + 0,587 \cdot G + 0,114 \cdot B, \tag{14}$$

$$Cb = -0,1687 \cdot R - 0,3313 \cdot G + 0,5 \cdot B + 128, \tag{15}$$

$$Cr = 0,5 \cdot R - 0,4187 \cdot G - 0,0813 \cdot B + 128. \tag{16}$$

Výše uvedené rovnice vracejí hodnoty v rozsahu $0 - 255$. Z důvodů správného výpočtu diskrétní kosinové transformace je nutné tento rozsah posunout do nového rozsahu od -128 do 128. To lze provést odečtením hodnoty 128 od jednotlivých složek. Upravené rovnice vypadají následovně:

$$Y = 0,299 \cdot R + 0,587 \cdot G + 0,114 \cdot B - 128, \tag{17}$$

$$Cb = -0,1687 \cdot R - 0,3313 \cdot G + 0,5 \cdot B, \tag{18}$$

$$Cr = 0,5 \cdot R - 0,4187 \cdot G - 0,0813 \cdot B. \tag{19}$$

```
function vystup = RGB_YCBCR(vstup)
vstup = double(vstup)//prevedeme na datovy typ double (kvuli nasobeni desetinnymi cisly)
[radky, sloupce] = size(vstup) //zjisteni poctu radku a sloupcu ze vstupu (obrazek)
//vytvoreni matice o velikosti radku a sloupcu ze vstupu (o trech kanalech RGB)
vystup = zeros(radky,sloupce,3)
//vypocet slozky Y
vystup(:, :, 1) = 0.299 * vstup(:, :, 1) + 0.587 * vstup(:, :, 2) + 0.114 * vstup(:, :, 3);
vystup(:, :, 2) = - 0.1687 * vstup(:, :, 1) - 0.3313 * vstup(:, :, 2) + 0.5 * vstup(:, :, 3) +
    128;//vypocet slozky Cb
vystup(:, :, 3) = 0.5 * vstup(:, :, 1) - 0.4187 * vstup(:, :, 2) - 0.0813 * vstup(:, :, 3) +
    128;//vypocet slozky Cr
endfunction
```

Výpis 4: Implementace funkce pro převod z barevného prostoru RGB do YCbCr.

Obrázek 20: Ukázka zobrazení všech složek z prostoru YCbCr.

Na obrázku 20 můžeme vidět převod z RGB prostoru do YCbCr prostoru. V levém horním rohu je vstupní obraz v RGB. Po převodu do YCbCr zde máme složku Y (vpravo nahoře), složku Cb (vlevo dole) a složku Cr (vpravo dole).

Pro převod z RGB do YCbCr je také možné použít funkci `rgb2ycbcr()` z balíčku IPCV, je však třeba dbát na to, že funkce vrací obraz ve formátu YCrCb (složky Cr a Cb jsou přehozeny).

3.2 Dopředná DCT

V předchozí kapitole 2 jsme si připomněli rozvoj funkce ve Fourierovu řadu a diskrétní Fourierovu transformaci. Také jsme uvedli jednorozměrné a dvourozměrné diskrétní kosinové transformace. Diskrétních kosinových transformací existuje několik typů, přesněji 8 standardních variant. V literatuře jsou označeny římskými číslicemi jako DCT I až DCT VIII [10].

Běžně se používají 4 varianty DCT. Nejpoužívanější variantou je DCT II. Tento typ je používán pro většinu praktických úloh. Dvourozměrná diskrétní kosinová transformace typu II (2D DCT) se používá například v souborových formátech. V našem případě se jedná o stan-

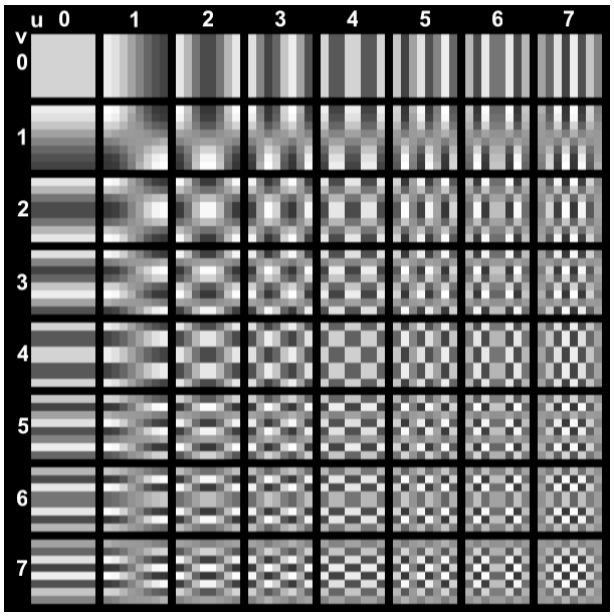
dard JPEG. Dále se používá ve formátech, které jsou určeny pro přenos videa. Příkladem jsou standardy MPEG 1, MPEG 2, MPEG 4. Výhody diskrétní kosinové transformace:

- Typický obrázek se spojitými přechody barev má vizuálně nejdůležitější informace o obrázku koncentrované v malém počtu kosinových funkcí o nízkých frekvencí.
- Pro zpětnou transformaci se využívá stejného postupu, protože u DCT jsou frekvenční koeficienty reálné.

Existují však i jiné transformace, které by byly vhodné pro některé úlohy. Příkladem může být vlnková transformace (Wavelet Transform) nebo KLT (Karhunen – Loeve Transform). Ty však z důvodů obchodních, patentových, nebo i implementačních prozatím DCT v praxi nepřekonaly.

Dříve, než se aplikuje DCT, se pro každou složku (Y, Cb, Cr) provádí rozdělení do bloků 8×8 pixelů. Při rozdělování obrázku na dané bloky se nemusí celý obrázek načítat do paměti, ale je dostačující osm aktuálních obrazových řádků. To je výhodné u specializovaných čipů, které se používají v kamerách nebo digitálních fotoaparátech. Pokud nastane situace, kdy šířka nebo výška není dělitelná osmi, je nutné doplnit zbývající hodnoty. Hodnoty by měly být doplněny tak, aby se do co největší míry snížil podíl energie ve vyšších frekvencích. To obecně vede k lepšímu komprimačnímu poměru.

Po DCT dostaneme 64 koeficientů. Ty si můžeme označit souřadnicemi (u, v), kde $u, v = 0, 1, \ldots 7$. První koeficient $(0, 0)$ je označen jako stejnosměrná složka (určuje průměrnou hodnotu v původním bloku). Ostatní koeficienty chápeme jako číslo, které určuje, kolikrát se daná frekvence vyskytuje v obrázku v daném směru (vertikálním nebo horizontálním).



Obrázek 21: 64 základních funkcí DCT pro blok 8×8.

Na obrázku 21 se v levé horní části vyskytují koeficienty, které mají nízké frekvence. Ty jsou pro lidské oko citlivější na vnímání. V pravé dolní části se vyskytují koeficienty, které mají vysoké frekvence. Běžný obrázek se spojitým rozložením hodnot obsahuje málo vysokofrekvenčních informací, z čehož plyne, že koeficienty o vyšších frekvencích budou nabývat menších hodnot.

```
function vystup = DCT_rovnice(vstup)
    [radky,sloupce] = size(vstup); //zjisteni vysky a sirky vstupu (obrazku)
    c = 1 / sqrt(2);
    dc = 2 / sqrt(sloupce * radky);

    for u = 0 : radky - 1 //prochazeni vsech radku daneho vstupu
        for v = 0 : sloupce - 1 //prochazeni vsech sloupcu daneho vstupu
            suma = 0; //suma nad vsemi pixely v bloku

            //pro kazdy pixel (u,v) se projde cely vstup znovu
            for m = 0 : radky - 1 //prochazeni vsech radku daneho vstupu
                for n = 0 : sloupce - 1 //prochazeni vsech sloupcu daneho vstupu
                  //ziskani hodnoty pixelu na souradnicich (m,n), zde se odecte hodnota 128 kvuli
                      spravnemu vypoctu DCT
                    hodnota_pixelu = double(vstup(m + 1,n + 1) - 128);
                    cos1 = cos(((2 * m + 1) * u * %pi)/(2 * radky));//cast rovnice DCT
                    cos2 = cos(((2 * n + 1) * v * %pi)/(2 * sloupce));//cast rovnice DCT
                    suma = suma + (hodnota_pixelu * cos1 * cos2);//cela rovnice DCT
                end
            end

            c_u = 1;//koeficient c_u
            c_v = 1;// koeficient c_v

            if u == 0 then //podminka pokud bude u = 1 do c_u se ulozi 1/sqrt(2) jinak 1
                c_u = c;
            end

            if v == 0 then //podminka pokud bude v = 1 do c_v se ulozi 1/sqrt(2) jinak 1
                c_v = c;
            end
            //do vystupu se ulozi vypocet cele rovnice
            vystup(u + 1,v + 1) = dc * c_u * c_v * suma;
        end
    end
endfunction
```

Výpis 5: Implementace rovnice pro DCT transformaci.

```
function vystup = DCT(vstup)
Y = vstup(:,:,1); //jasova slozka Y
[radky,sloupce]=size(vstup)//ziskani pocet radku a sloupcu
vystup = zeros(radky,sloupce)//vytvoreni matice
//prochazeni obrazku po blocich 8x8
for y = 1 : 8 : radky - 7//prochazeni vsech radku po 8 pixelech
   for x = 1 : 8 : sloupce - 7 //prochazeni vsech sloupcu po 8 pixelech
      //vlozeni DCT bloku na vystup
      vystup(y : y + 7,x : x + 7) = DCT_rovnice(Y(y : y + 7,x : x + 7));
   end
end
endfunction
```

Výpis 6: Implementace DCT transformace.

3.3 Kvantizace DCT koeficientů

Nejdůležitějším krokem pro získání vynikajících kompresních výsledků s algoritmy JPEG je kvantování koeficientů po diskrétní kosinové transformaci [11]. Všechny koeficienty po spektrální analýze jsou vyděleny kvantizační maticí Q (jejími koeficienty) a zaokrouhleny na celé číslo podle rovnice 20:

$$F^Q(u,v) = \text{Round}\left[\frac{F(u,v)}{Q(u,v)}\right], \tag{20}$$

kde $F(u,v)$ je matice barev pixelů (jedna ze složek Y, Cb, Cr) a $Q(u,v)$ je kvantizační matice. Rozměr kvantizační tabulky je 8×8. Každá z 64 frekvencí má svůj vlastní kvantizační koeficient.

Jsou zavedeny oddělené kvantizační tabulky pro jasové a barvonosné složky. To dovoluje využít rozdílnou citlivost oka na jas a barvu. Konkrétní kvantizační tabulky, jak pro jasovou složku, tak pro barvonosné složky, nejsou standardem JPEG předepsány. Je tedy teoreticky možné si zvolit tabulky vlastní. Tyto tabulky by měly být vytvořeny s ohledem na charakter obrazu.

I když kvantizační tabulky nejsou standardem, norma uvádí tabulky, které byly experimentálně stanoveny v rámci doporučení CCIR-601. Tyto tabulky jsou v mnoha případech používány.

Tabulka 1: Kvantizační tabulka pro jasovou složku Y (vlevo) a pro barvonosné složky Cb a Cr (vpravo).

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

V mé diplomové práci používám doporučené tabulky 1. Hodnoty, které jsou v tabulkách, určují v jaké kvalitě bude obrázek uložen. Čím větší je hodnota v tabulce, tím méně přesný bude koeficient. Kvantizace tedy probíhá tak, že se jednotlivé položky vydělí odpovídajícími hodnotami v kvantizační tabulce. Výsledkem je blok kvantizačních koeficientů.

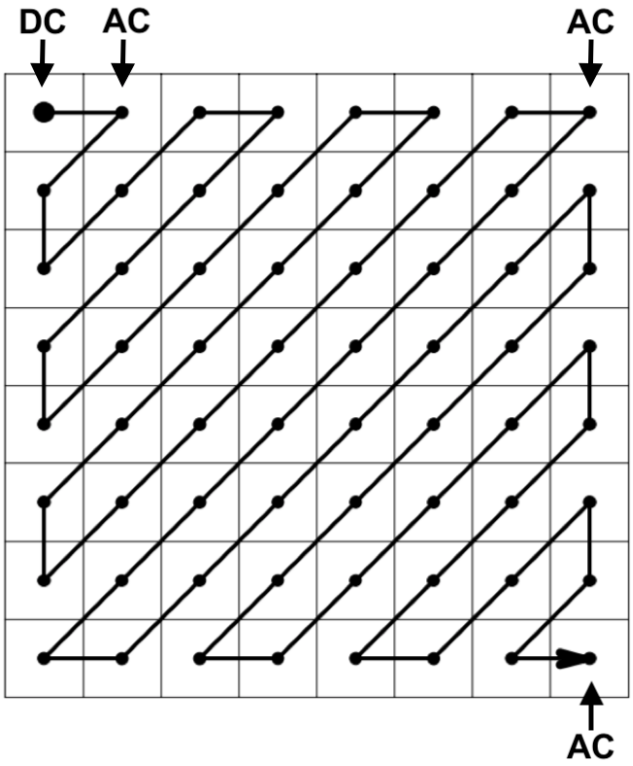
```
function vystup = Kvantovani(vstup)
    matice_Y = [
        16,11,10,16,24,40,51,61;//kvantizacni matice pro jasovou slozku
        12,12,14,19,26,58,60,55;
        14,13,16,24,40,57,69,56;
        14,17,22,29,51,87,80,62;
        18,22,37,56,68,109,103,77;
        24,35,55,64,81,104,113,92;
        49,64,78,87,103,121,120,101;
        72,92,95,98,112,100,103,99];

    Y = vstup(:,:,1); //jasova slozka Y
    [radky,sloupce]=size(vstup)//ziskani pocet radku a sloupcu
    vystup = zeros(radky,sloupce)//vytvoreni matice

    for y = 1 : 8 : radky - 7
        for x = 1 : 8 : sloupce - 7
            // Vsechny koeficienty jsou vydelene kvantizacni matici (jejimi koeficienty) a
                zaokrouhleny na cele cislo
            vystup(y : y + 7,x : x + 7) = round(Y(y : y + 7,x : x + 7) ./ matice_Y)
        end
    end
endfunction
```

Výpis 7: Implementace kvantizace DCT koeficientů.

3.4 Kódování

Po kvantizaci DCT koeficientů se dále provede předposlední krok JPEG komprese, a tím je kódování [12]. Jedná se o bezztrátové zakódování kvantových koeficientů DCT (všech jednotlivých bloků). Kódování se provádí dvěma způsoby, a to buď Huffmanovým, nebo aritmetickým kódováním.

Po kvantizaci se koeficient DC (stejnosměrná složka) a 63 koeficientů AC (střídavá složka) upravuje pro kódování. Hodnoty matice 8×8 jsou uspořádány do speciální posloupnosti, které se říká *zig-zag posloupnost*. Matice se prochází postupně, jak je znázorněno na obrázku 22. Přeorganizování způsobí, že hodnoty odpovídající nejnižším frekvencím se dostávají z levé horní části matice DCT na začátek posloupnosti zig-zag. Hodnoty, které odpovídají vyšším frekvencím, vytvářejí v posloupnosti zig-zag běh za sebou následujících nul. Nalezení nenulové hodnoty vede k vytvoření nové položky v zig-zag posloupnosti.



Obrázek 22: Zig-zag posloupnost.

```
function vystup = zigZagIndexy(velikostMatice)
//tato funkce slouzi pro ziskani indexu pri zig-zag posloupnosti, bude pouzita ve funkci
    Sekvence, kde se vytvari intermedialni sekvence
// pri pruchodu matice se prehodi bud sloupec nebo radek
prehodit_sloupec = %t; //nastaveni prehodit_sloupec na true (prvni se prehodi sloupec)
prehodit_radek = %f; //nastaveni prehodit_radek na false
vystup = list(); // vystupem bude vektor

// Tato smycka je pro horni diagonalu matice
for i = (2 : velikostMatice) //prochazime matici od AC koeficientu
   radek = (1 : i); //generuje pozice od 1 do i pro radek
   sloupec = (1 : i);//generuje pozice od 1 do i pro sloupec
   //pokud prehodit_sloupec je true, poradi ve sloupci se prohodi (poradi 1,2,3 bude 3,2,1)
   if prehodit_sloupec
      sloupec = sloupec(:,$ : -1 : 1);
      prehodit_radek = %t; //nyni se nastavi prehodit_radek na true
      prehodit_sloupec = %f; //nyni se nastavi prehodit_sloupec na false
   //pokud je prehodit_radek true, poradi v~radku se prohodi (poradi 1,2,3 bude 3,2,1)
   elseif prehodit_radek
      radek = radek(:,$ : -1 : 1);
      prehodit_radek = %f; //opet se nastavi prehodit_radek na false
      prehodit_sloupec = %t; //prehodit_sloupec se nastavi na true
   end
   //projdou se vsechny zpracovane indexy a~ulozi se na konec vystupu
   for j = (1 : length(radek))
      vystup($ + 1) = [radek(j),sloupec(j)]
   end
end

// Tato smycka je pro dolni diagonalu matice
for i = (2 : velikostMatice) //opet se projizdi matce od AC koeficientu
   radek = (i : velikostMatice); //generuji se pozice od i do konce matice pro radek
   sloupec = (i : velikostMatice); // generuji se pozice od i do konce matice pro sloupec
   if prehodit_sloupec
      sloupec = sloupec(:,$ : -1 :1); prehodit_radek = %t; prehodit_sloupec = %f;
   elseif prehodit_radek
      radek = radek(:,$ : -1 : 1); prehodit_radek = %f; prehodit_sloupec = %t;
   end
   //opet se projdou vsechny zpracovane indexy a~ulozi se na konec vystupu
   for j = (1 : length(radek))
      vystup($ + 1) = [radek(j),sloupec(j)]
   end
end
endfunction
```

Výpis 8: Implementace získání indexů zig-zag posloupnosti.

Při průchodu zig-zag posloupností se tvoří intermediální sekvence. Položka v sekvenci má tvar dvojice *Symbol-1* a *Symbol-2*. Jednotlivé symboly se kódují odlišně. Popíšeme si nejdříve vytváření stejnosměrné složky.

Stejnosměrná složka se skládá ze *Symbol-1* a *Symbol-2*. *Symbol-1* obsahuje údaj *Size*. Tento údaj nám říká, kolik je potřeba bitů k reprezentaci hodnoty Amplitude. *Symbol-2* obsahuje údaj *Amplitude* (hodnota aktuálního bodu). DC koeficienty se vytvářejí diferenčně 21, protože mezi DC koeficienty sousedních bloků 8×8 je silná korelace.

$$DIFF = DC_i - DC_{(i-1)}. \tag{21}$$

Střídavé složky AC se skládají také ze *Symbol-1* a *Symbol-2*. *Symbol-1* obsahuje dvě informace. *První informací* je *Runlength*. To je číslo určující počet za sebou jdoucích nulových hodnot AC koeficientů, které předcházejí nenulovému AC koeficientu. Hodnota parametru je v rozmezí 0 až 15, tzn., že pro jeho reprezentaci potřebujeme 4 bity. Může nastat situace, kdy bude počet nul větší než 15, potom se zde zavádí hodnota *Symbol-1(15,0)*. Tato hodnota říká, že po průchodu patnácti nulovými hodnotami nebyla zatím nenulová hodnota matice nalezena. Hodnota *Symbol-1(0,0)* označuje konec bloku a slouží k ukončení každého bloku 8×8 bodů.

Druhou informací je *Size*. Size udává počet bitů, které jsou zapotřebí k reprezentaci hodnoty Amplitude. Počet bitů se pohybuje v rozmezí 0 až 10, což opět vyžaduje 4 bity pro jeho reprezentaci.

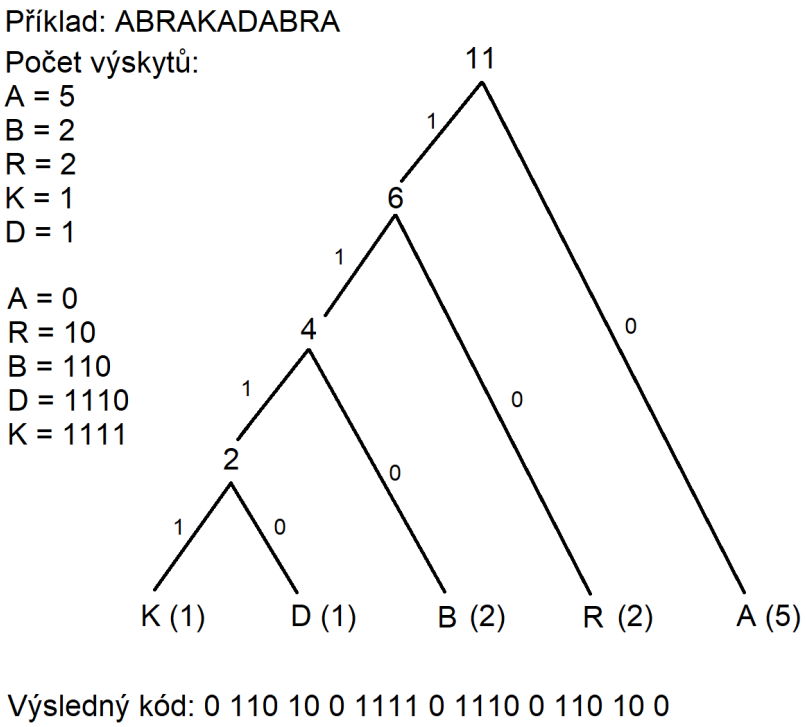
Symbol-2 obsahuje pouze jediný údaj *Amplitude*. Amplitude udává velikost nenulového koeficientu AC v rozmezí od −1023 do 1023. Takové rozmezí vyžaduje 10 bitů pro jeho reprezentaci.

Po vytvoření přechodné posloupnosti symbolů nastává poslední fáze komprese, kterou je zakódování dané sekvence. Zakódování provádíme, jak už bylo řečeno, dvěma způsoby.

První způsob je Huffmanovo kódování (obrázek 23). To probíhá tak, že využívá četnosti výskytů jednotlivých znaků. K často používaným znakům jsou přiřazeny krátké kódy a k méně používaným znakům jsou přiřazeny dlouhé kódy. Pro kódování se používá binární strom, který se tvoří od jednotlivých listů ke kořenu. Hrany stromu jsou ohodnoceny symboly 0 a 1. Uzly jsou ohodnoceny pravděpodobností výskytu daného znaku ze vstupní sekvence. Správně vytvořený Huffmanův kód má minimální délku a je jednoznačně dekódovatelný (jedná se o prefixový kód).

Postup při kódování:

1. Nejdříve je nutné vytvořit množinu četností a seřadit je od nejvyšší četnosti po nejnižší.
2. Dále se sečtou dvě nejnižší četnosti a jejich součet se považuje za jednu četnost.
3. Krok 2 se opakuje, dokud nebude vytvořen celý strom.
4. Jednotlivé četnosti se procházejí odspodu směrem nahoru.
5. Kódové kombinace jsou tvořeny od kořene směrem k listům.



Obrázek 23: Příklad Huffmanova kódování.

Samotné kódování komprese JPEG spočívá ve vyhledávání v tabulkách. Tabulka pro Huffmanovo kódování může nabývat libovolných hodnot, nebo může mít referenční hodnoty uvedené ve standardu. JPEG uvádí ve svém standardu základní tabulky. Jedná se o tabulky (DC a AC) pro jasovou složku Y a pro barvonosné složky Cb a Cr. V této diplomové práci jsou používány referenční tabulky, viz tabulka 2. Ta udává tabulky pro rozdíly v koeficientu DC, které byly vyvinuty z průměru statistiky velké sady snímků s přesností 8 bitů. Levá část je určená pro komponenty s jasem a pravá část je určená pro chrominanční komponenty. Přestože neexistují výchozí tabulky, mohou se tyto tabulky ukázat jako užitečné v mnoha aplikacích.

Tabulka 2: Tabulky (DC) pro jasovou složku Y (vlevo) a pro barvonosné složky Cb a Cr (vpravo).

Kategorie	Délka kódu	Kódové slovo
0	2	00
1	3	010
2	3	011
3	3	100
4	3	101
5	3	110
6	4	1110
7	5	11110
8	6	111110
9	7	1111110
10	8	11111110
11	9	111111110

Kategorie	Délka kódu	Kódové slovo
0	2	00
1	2	01
2	2	10
3	3	110
4	4	1110
5	5	11110
6	6	111110
7	7	1111110
8	8	11111110
9	9	111111110
10	10	1111111110
11	11	11111111110

Stejně jako existují tabulky pro DC koeficienty, existují i tabulky pro AC koeficienty, které byly také vyvinuty z průměrné statistiky velké sady snímků s přesností 8 bitů. Tyto tabulky 14 a 15 jsou uvedeny v příloze A a B, důvodem je velké množství dat.

```
function vystup = Sekvence(vstup)
[radky,sloupce] = size(vstup);//zjisteni vysky a sirky vstupu (obrazek)
Y = vstup(:,:,1);//ulozeni jasove slozky Y
mojeIndexy = zigZagIndexy(8);//obsahuje indexy matice bloku v zig-zag posloupnosti
vysledna_sekvence = list();//definovani vysledne intermedialni sekvence
predchozi_dc = 0;//hodnota predchozi dc slozky (prvni je vzdy 0)
for y = 1 : 8 : radky - 1//prochazeni po blocich
   for x = 1 : 8 : sloupce - 1
      pocet_nul = 0;//pocet nul v aktualni sekvenci
      aktualni_sekvence = list();//aktualni intermedialni sekvence

      //DC slozky
      aktualni_hodnota_dc = Y(y,x);//aktualni hodnota DC slozky
      vysledne_dc = aktualni_hodnota_dc - predchozi_dc;//rozdil DC slozek
      //na konec akt. sekvence se ulozi dvojice (pocet bitu k reprezentaci hodnoty,hodnota)
      aktualni_sekvence($ + 1) = [length(dec2bin(abs(vysledne_dc))),vysledne_dc];
      predchozi_dc = aktualni_hodnota_dc;// DIFF = DCi - DC(i-1)

      //AC slozky
      for i = (1 : length(mojeIndexy))//prochazeni listu souradnic zig-zag posloupnosti
         bunka = mojeIndexy(i);//ziskani souradnic na pozici i
         k = bunka(1) + y - 1;//posunuti souradnic y zig-zagu do aktualniho bloku
         l = bunka(2) + x - 1;//posunuti souradnic x zig-zagu do aktualniho bloku
         aktualni_hodnota_ac = Y(k,l);//hodnoty ac na aktualnich souradnicich zig-zagu
         if aktualni_hodnota_ac == 0 then //pokud nalezneme nulu, pocet nul navysime
            pocet_nul = pocet_nul + 1;
         else
         //zapiseme do sekvence dvojici (pocet nul,pocet bitu k reprezentaci hodnoty, hodnota)
         sekvence_bloku = [pocet_nul, length(dec2bin(abs(aktualni_hodnota_ac))),
             aktualni_hodnota_ac];
         if pocet_nul > 15 then//muze nabyvat max 15, jinak se zavadi spec. dvojice
            sekvence_bloku(1) = 15;
            sekvence_bloku(2) = 0;
         end
         //na konci aktualni sekvence se zapise AC slozka
         aktualni_sekvence($ + 1) = sekvence_bloku;
         pocet_nul = 0; //vynuluje se pocet nul
      end

      aktualni_sekvence($ + 1) = [0, 0];// na konec se prida priznak konce bloku (0,0)
      //aktualni sekvence se ulozi do vyslednface sekvence
      vysledna_sekvence($ + 1) = aktualni_sekvence;
   end
end
vystup = vysledna_sekvence;//vystup jsou sekvence jednotlivych bloku za sebou
endfunction
```

Výpis 9: Implementace vytvoření intermediální sekvence.

```
function vystup = Kodovani_symbol_2(vstup)
//pokud je vstup vetsi nebo roven nule, zapise se cislo v binarni soustave
if (vstup >= 0) then
   vystup = dec2bin(vstup)
//provede se jednotkovy doplnek, tzn. z nulovych bitu udela jednotkove a naopak
else
   vystup = strsubst(dec2bin(abs(vstup)),'1','_');//nejdrive nahradime jednicky podtrzitky
   vystup = strsubst(vystup,'0','1');//pote nuly jednickami
   vystup = strsubst(vystup,'_','0');//nakonec podtrzitka nulami
end
endfunction
```

Výpis 10: Implementace kódování Symbol-2.

```
function vystup = Kodovani(vstup)
DC_koeficienty = list('00', '010', '011', '100', '101', '110', '1110', '11110', '111110', '
    1111110', '11111110', '111111110');
AC_koeficienty = list();
AC_koeficienty(1) = list('1010', '00', '01', '100', '1011', '11010', '1111000', '11111000', '
    1111110110', '1111111110000010', '1111111110000011');
AC_koeficienty(2) = list('0','1100', '11011', '1111001', '111110110', '11111110110', '
    1111111110000100', '1111111110000101', '1111111110000110', '1111111110000111', '
    1111111110001000');
AC_koeficienty(3) = list('0','11100', '11111001', '1111110111', '111111110100', '
    1111111110001001', '1111111110001010', '1111111110001011', '1111111110001100', '
    1111111110001101', '1111111110001110');
AC_koeficienty(4) = list('0','111010', '111110111', '111111110101', '1111111110001111', '
    1111111110010000', '1111111110010001', '1111111110010010', '1111111110010011', '
    1111111110010100', '1111111110010101');
//dale jsou koeficienty ulozeny az do indexu 16, ale pro velke mnozstvi dat je pro ukazku
    zobrazeno jen male mnozstvi
vysledny_retezec = '';//vysledny retezec pro zakodovani
for i = 1 : length(vstup)//prochazeni vstupni sekvence
blok = vstup(i);//ziskani sekvence pro dany blok
vysledny_retezec = vysledny_retezec + DC_koeficienty(blok(1)(1) + 1) + Kodovani_symbol_2(blok
    (1)(2));//zakodovani DC slozky a nasledne ulozeni

for j = 2 : length(blok) - 1//prochazeni sekvence pro dany blok od 2 koeficientu
vysledny_retezec = vysledny_retezec + AC_koeficienty(blok(j)(1) + 1)(blok(j)(2) + 1) +
    Kodovani_symbol_2(blok(j)(3));//zakodovani AC slozky a nasledne ulozeni
end
vysledny_retezec = vysledny_retezec + AC_koeficienty(blok(length(blok))(1) + 1)(blok(length(
    blok))(2) + 1);//zakodovani priznaku konce bloku
end
vystup = vysledny_retezec;//nasledne ulozeni zakodovane sekvence do vystupu
endfunction
```

Výpis 11: Implementace kódování intermediální sekvence.

Druhým způsobem kódování je aritmetické kódování. Používá metody, během nichž je celá sekvence zakódována do reálného čísla v rozsahu (0,1). Tato metoda má řadu různých variant, jak převést výsledné reálné číslo do digitální podoby. V případě JPEGu nejde o typické znaky, jako je například ASCII, ale jedná se o čísla kategorií, sekvenci bytů nebo dvojici či trojici znaků. Aritmetické kódování nebylo obsahem této diplomové práce, a proto zde nebude dále rozvedeno.

Huffmanovo či arimetické kódování představuje nejdůležitější část JPEG komprese. Při špatném návrhu by veškerá dosavadní práce přišla vniveč (transformace barev, DCT, kvantizace DCT koeficientů a vytvoření intermediální sekvence). Tyto kroky se provádějí kvůli snadno zpracovatelnému výsledku. Výsledkem by měla být sekvence bitů, kterou snadno zpracuje právě Huffmanovo nebo aritmetické kódování.

4 Ukládání výsledného JPEG souboru

Po zvládnutí všech předchozích kroků následuje ukládání vytvořeného obrázku do formátu JPEG [6],[7]. Ukládání se skládá z několika částí. Jedná se o řadu značek, které jsou zapisovány po sobě v hlavičce souboru. Každé značce předchází 1 byte (0xFF). V této kapitole se budeme zabývat značkami, které jsou uvedeny v tabulce 3. Všechny níže uvedené příklady se zabývají pouze jasovou složkou obrazu. V případě barevného obrazu je nutné některé části značek hlavičky upravit tak, aby odpovídaly pravidlům popsaným v tabulkách.

Tabulka 3: Značky hlavičky JPEG souboru.

Název značky	Identifikátor značky	Popis značky
SOI	0xD8	Začátek obrázku
APP0	0xE0	JFIF aplikační segment
DQT	0xDB	Kvantizační tabulka
DHT	0xC4	Tabulka pro Huffmanovo kódování
SOF0	0xC0	Začátek rámce SOF0
SOS	0xDA	Začátek obrazových dat
Komprimovaná data		
EOI	0xD9	Konec obrázku

4.1 SOI - Start of Image

Hlavička souboru začíná značkou SOI (Start of Image), která se značí 0xD8. Jak bylo řečeno, každé značce předchází 1 byte (0xFF) [17].

Tabulka 4: Značka SOI v hlavičce souboru.

Název	Popis
Identifikátor značky	Identifikace SOI: 0xFF, 0xD8

```
//otevreni souboru, do ktereho budeme zapisovat hlavicky a komprimovana data
soubor = mopen(jmeno_souboru,'wb');
mput(hex2dec('FFD8'),'usb');//identifikator zacatku obrazku
```

Výpis 12: Implementace značky SOI (Star of image).

4.2 APP0 - JFIF application segment

Po značce SOI následuje značka APP0. Tato značka ukazuje JFIF segment, který obsahuje identifikaci JFIF souboru.

Tabulka 5: Značka APP0 v hlavičce souboru.

Název	Velikost [B]	Popis
Identifikátor značky	2	Identifikace APP0: 0xFF, 0xE0
Délka	2	Musí být ≥ 16
Identifikátor souboru	5	Identifikace JFIF souboru $'JFIF'\#0$ (0x4A, 0x46, 0x49, 0x46, 0x00)
Hlavní číslo verze	1	1
Vedlejší číslo verze	1	V rozsahu 0 až 2
Jednotky hustoty x/y	1	0 = žádné jednotky, specifikují se poměry stran 1 = hustota je bod/palec 2 = hostota je bod/cm
Hustota x	2	$\neq 0$
Hustota y	2	$\neq 0$
Šířka náhledu	1	–
Výška náhledu	1	–
Data náhledu	n	$n =$ šířka $\cdot$ výška $\cdot 3$

Pokud neexistuje žádný $'JFIF'\#0$, nebo je délka < 16, pak pravděpodobně není segmentem JFIF a identifikátor souboru by měl být ignorován. Základní nastavení měřítka: jednotky hustoty $x/y = 0$, hustota $x = 1$, hustota $y = 1$. To znamená, že poměr stran je $1 : 1$ (rovnoměrné měřítko).

Soubory JFIF včetně náhledů jsou velmi vzácné. Náhledy jsou obvykle ignorovány. Pokud není žádný náhled, pak šířka = 0 a výška = 0. Jestliže délka neodpovídá velikosti náhledů, může se vyskytnout varování.

```
mput(hex2dec('FFE0'),'usb');//identifikator znacky
mput(16,'usb');//delka = 16
mput(hex2dec('4A464946'),'uib');//identifikator souboru
mput(hex2dec('00'),'ucb');//Terminal
mput(hex2dec('0101'),'usb');//JFIF verze
mput(hex2dec('00'),'ucb');//jednotky hustoty x a y
mput(1,'usb');// hustota x
mput(1,'usb');//hustota y
mput(hex2dec('00'),'ucb');//xtn
mput(hex2dec('00'),'ucb');//ytn
```

Výpis 13: Implementace značky APP0 - JFIF segment.

4.3 DQT - Quantization Table

Další částí hlavičky je značka definující kvantizační tabulku. Může být zaznamenáno více druhů tabulek, až tři druhy s přesností 8 bitů. Všechny tabulky kvantizace jsou definovány v jednom segmentu DQT. Nahrávání více značek DQT není povoleno. Kvantizační tabulka může nabývat libovolných hodnot, nebo může mít referenční hodnoty standardu. JPEG uvádí ve svém standardu základní tabulky. DQT, DHT a SOF se mohou řadit v libovolném pořadí, ale budou zaznamenány po APP0 a před SOS.

Tabulka 6: Značka DQT v hlavičce souboru.

Název	Velikost [B]	Popis
Identifikátor značky	2	Identifikace DQT: 0xFF, 0xDB
Délka	2	Udává velikost QT
QT informace	1	bity 0 až 3: číslo QT bity 4 až 7: přesnost QT, 0 = 8 bitů, jinak 16 bitů
Data	n	Udává QT hodnoty, $n = 64 \cdot (\text{přesnost} + 1)$

Jeden segment DQT může obsahovat více QT (pro jasovou složku a barvonosné složky), z nichž každý má svůj vlastní byte informací. Pro přesnost = 1 (16 bitů) je pořadí bitů od nejmenšího po největší pro každé z 64 slov.

```
matice_Y = [
16,11,10,16,24,40,51,61;
12,12,14,19,26,58,60,55;
14,13,16,24,40,57,69,56;
14,17,22,29,51,87,80,62;
18,22,37,56,68,109,103,77;
24,35,55,64,81,104,113,92;
```

```
49,64,78,87,103,121,120,101;
72,92,95,98,112,100,103,99]; //kvantizacni tabulka

mput(hex2dec('FFDB'),'usb');//identifikator znacky
mput(67,'usb');//delka kvantizacni tabulky, identifikatoru znacky a qt informace
mput(hex2dec('00'),'ucb');//QT informace 0 = tabulka, 0 = 8 bit
for s = 1 : 8
   for r = 1 : 8
      mput(matice_Y(s,r),'ucb');//zapisovani kvantizacni tabulky do souboru po jedno bytu
   end
end
```

Výpis 14: Implementace značky DQT (Quantization Table).

4.4 DHT - Huffman Table

V této části hlavičky se ukládá tabulka pro Huffmanovo kódování, která se skládá z DC a AC složky [13],[14]. Každá komponenta (jasová složka Y nebo barvonosné složky Cb a Cr) musí být mapována na jednu tabulku. Tabulky jsou definované v DHT. Všechny tabulky budou zaznamenány v jednom segmentu DHT. Jeden segment DHT může obsahovat více HT, každý s vlastním informačním bytem. Nahrávání více značek DHT není povoleno. Tabulka pro Huffmanovo kódování může buď nabývat libovolných hodnot, nebo může mít referenční hodnoty standardu. JPEG uvádí ve svém standardu základní tabulky.

Tabulka 7: Značka DHT v hlavičce souboru.

Název	Velikost [B]	Popis
Identifikátor značky	2	Identifikace DHT: 0xFF, 0xC4
Délka	2	Specifikuje velikost HT
HT informace	1	bity 0 až 3: číslo HT bit 4: typ HT, 0 = DC tabulka, 1 = AC tabulka bity 5 až 7: musí být 0, nepoužívá se
Počet symbolů	16	Počet symbolů s kódem délky 1 až 16, suma délek by měla být rovna celkovému počtu kódu, musí být ≤ 256
Symboly	n	Tabulka obsahující symboly v pořadí zvětšující se délky kódu (n = celkový počet kódů)

```
// DC slozka
// DC koeficienty
DC_koeficienty = ['00' '01' '02' '03' '04' '05' '06' '07' '08' '09' '0A' '0B'];
DC_delka_kodu = ['00' '01' '05' '01' '01' '01' '01' '01' '01' '00' '00' '00' '00' '00' '00' '00'
    ]; // delky kodu
// identifikator znacky, ht informace ,velikost dc koeficientu a velikost delek kodu
delka_segmentu_DHT = 3 + size(DC_koeficienty)(2) + size(DC_delka_kodu)(2);
mput(hex2dec('FFC4'),'usb');//identifikator znacky
mput(delka_segmentu_DHT,'usb');//delka segmentu DHT
mput(hex2dec('00'),'ucb');//HT informace 0 = DC tabulka, 0 = musi byt pouzita

for m = 1 : size(DC_delka_kodu)(2)
mput(hex2dec(DC_delka_kodu(m)),'ucb');//zapisovani DC slozky (delku kodu)
end

for j = 1 : size(DC_koeficienty)(2)
mput(hex2dec(DC_koeficienty(j)),'ucb'); //zapisovani DC slozky (dc koeficienty)
end
//AC slozka
AC_delka_kodu = ['00' '02' '01' '03' '03' '02' '04' '03' '05' '05' '04' '04' '00' '00' '01' '7D'
    ]; // AC slozka (delka kodu)
AC_koeficienty = [
'01' '02' '03' '00' '04' '11' '05' '12' '21' '31' '41' '06' '13' '51' '61' '07' ...
'22' '71' '14' '32' '81' '91' 'A1' '08' '23' '42' 'B1' 'C1' '15' '52' 'D1' 'F0' ...
'24' '33' '62' '72' '82' '09' '0A' '16' '17' '18' '19' '1A' '25' '26' '27' '28' ...
'29' '2A' '34' '35' '36' '37' '38' '39' '3A' '43' '44' '45' '46' '47' '48' '49' ...
'4A' '53' '54' '55' '56' '57' '58' '59' '5A' '63' '64' '65' '66' '67' '68' '69' ...
'6A' '73' '74' '75' '76' '77' '78' '79' '7A' '83' '84' '85' '86' '87' '88' '89' ...
'8A' '92' '93' '94' '95' '96' '97' '98' '99' '9A' 'A2' 'A3' 'A4' 'A5' 'A6' 'A7' ...
'A8' 'A9' 'AA' 'B2' 'B3' 'B4' 'B5' 'B6' 'B7' 'B8' 'B9' 'BA' 'C2' 'C3' 'C4' 'C5' ...
'C6' 'C7' 'C8' 'C9' 'CA' 'D2' 'D3' 'D4' 'D5' 'D6' 'D7' 'D8' 'D9' 'DA' 'E1' 'E2' ...
'E3' 'E4' 'E5' 'E6' 'E7' 'E8' 'E9' 'EA' 'F1' 'F2' 'F3' 'F4' 'F5' 'F6' 'F7' 'F8' ...
'F9' 'FA'
] //AC koeficienty
// identifikator znacky, ht informace, velikost AC koeficientu, velikost AC slozky (delky kodu)
delka_segmentu_DHT = 3 + size(AC_koeficienty)(2) + size(AC_delka_kodu)(2);
mput(hex2dec('FFC4'),'usb'); //identifikator znacky
mput(delka_segmentu_DHT,'usb'); //delka segmentu DHT
mput(bin2dec('00010000'),'ucb');//HT informace 1 = AC tabulka, 0 = musi byt pouzita

for k = 1 : size(AC_delka_kodu)(2)
mput(hex2dec(AC_delka_kodu(k)),'ucb');//zapisy do souboru
end

for p = 1 : size(AC_koeficienty)(2)
mput(hex2dec(AC_koeficienty(p)),'ucb');//zapisy do souboru
end
```

Výpis 15: Implementace značky DHT (Huffman Table).

4.5 SOF0 - Start of Frame

Další částí hlavičky je značka SOF0 (Start of Frame). Jedná se o značkovací kód označující začátek segmentu rámce a udávající různé parametry pro tento rámec.

Tabulka 8: Značka SOF0 v hlavičce souboru.

Název	Velikost [B]	Popis
Identifikátor značky	2	Identifikace SOF0: 0xFF, 0xC0
Délka	2	Tato hodnota se rovná $8 + \text{komponenta} \cdot 3$
Přesnost dat	1	Platí bit/vzorek, obvykle 8 (12 a 16 není doporučeno)
Výška obrázku	2	Musí být > 0
Šířka obrázku	2	Musí být > 0
Počet komponent	1	Obvykle 1 = grey scaled, 3 = color YcbCr nebo YIQ 4 = color CMYK
Jednotlivé komponenty	3	ID: 1 = Y, 2 = Cb, 3 = Cr, 4 = I, 5 = Q (1B) Vzorkovací faktory: bity 0 až 3 vertikálně, 4 až 7 horizontálně (1B) Číslo kvantizační tabulky (1B)

JFIF používá 1 komponentu (Y, greyscaled) nebo 3 komponenty (YCbCr, někdy zvané YUV).

```
mput(hex2dec('FFC0'),'usb');//identifikator znacky
mput(11,'usb');//delka hlavicky
mput(8,'ucb');//presnost dat
mput(radky,'usb');//vyska obrazku
mput(sloupce,'usb');//sirka obrazku
mput(1,'ucb');//cislo komponenty 1 = jasova slozka
mput(1,'ucb');//ID = 1 (jasova slozka)
mput(hex2dec('22'),'ucb');// vzorkovaci faktory
mput(0,'ucb');//cislo kvantizacni tabulky
```

Výpis 16: Implementace značky SOF0 (Star of Frame).

4.6 SOS - Start of Scan

SOS je značkovací kód s různými parametry týkajícími se kontroly. Po SOS segmentu musí následovat komprimovaná data.

Tabulka 9: Značka SOS v hlavičce souboru.

Název	Velikost [B]	Popis
Identifikátor značky	2	Identifikace SOS: 0xFF, 0xDA
Délka	2	Musí se rovnat $6 + 2 \cdot$ počet komponent
Počet komponent	1	Musí být ≥ 1 a ≤ 4, obvykle 1 nebo 3
Jednotlivé komponenty	2	ID: 1=Y, 2=Cb, 3=Cr, 4=I, 5=Q (1 byte) Huffmanova tabulka používá: bity 1 až 3: AC tabulka bity 4 až 7: DC tabulka
Ignorovatelné byty	3	3B se musí přeskočit

```
mput(hex2dec('FFDA'),'usb');//identifikator znacky
mput(8,'usb');//delka
mput(1,'ucb');//pocet komponent, ktere jsou pouzity
mput(hex2dec('01'),'ucb');//ID = 1 (jasova slozka)
mput(hex2dec('00'),'ucb');//ID huffmanovy tabulky
mput(hex2dec('00'),'ucb');//ignorovatelne byty
mput(hex2dec('3F'),'ucb');//ignorovatelne byty
mput(hex2dec('00'),'ucb');//ignorovatelne byty
```

Výpis 17: Implementace značky SOS (Start of Scan).

4.7 Komprimovaná data

Další částí hlavičky jsou komprimovaná data. To je Huffmanem zakódovaný bytový datový blok v požadovaném poměru Y:Cb:Cr. EoB (konec bloku) je vložen do každého bloku, jestliže poslední hodnotou jeho koeficientu DCT je 0. Pokud se v komprimovaných obrazových datech vyskytne 0xFF, doplní se za něj nulový byte 0x00.

```
delka_retezce = length(vstup); //nacteme si data
//zjisteni, zda jsou data delitelna osmi (1 byte), a pripadne ulozeni zbytku po celociselnem
    deleni
zbytek = modulo(delka_retezce,8);
//ziskani casti z konce dat, ktera se nevesla do 1 bytu
zbytek2 = part(vstup,delka_retezce - zbytek + 1 : delka_retezce);
for z = zbytek : 7 //doplneni zbytku dat do delky 1 bytu
zbytek2 = zbytek2 + '1' //doplni se jednickama
end
//nahrazeni puvodniho zbytku zbytkem2, ktery je doplnen jednickami
vstup = part(vstup,1 : delka_retezce - zbytek) + zbytek2;

for n = 1 : 8 : delka_retezce - zbytek
//zapis jednotlivych bytu komprimonavych dat do souboru
mput(bin2dec(part(vstup,n : n + 7)),'ucb');
//jestlize se vyskytne 0xff musi nasledovat nulovy byte
if part(vstup,n : n + 7) == '11111111' then
mput(0,'ucb');//zapsani nulove bytu do souboru
end
end
```

Výpis 18: Implementace komprimovaných dat.

4.8 EOI - End of Image

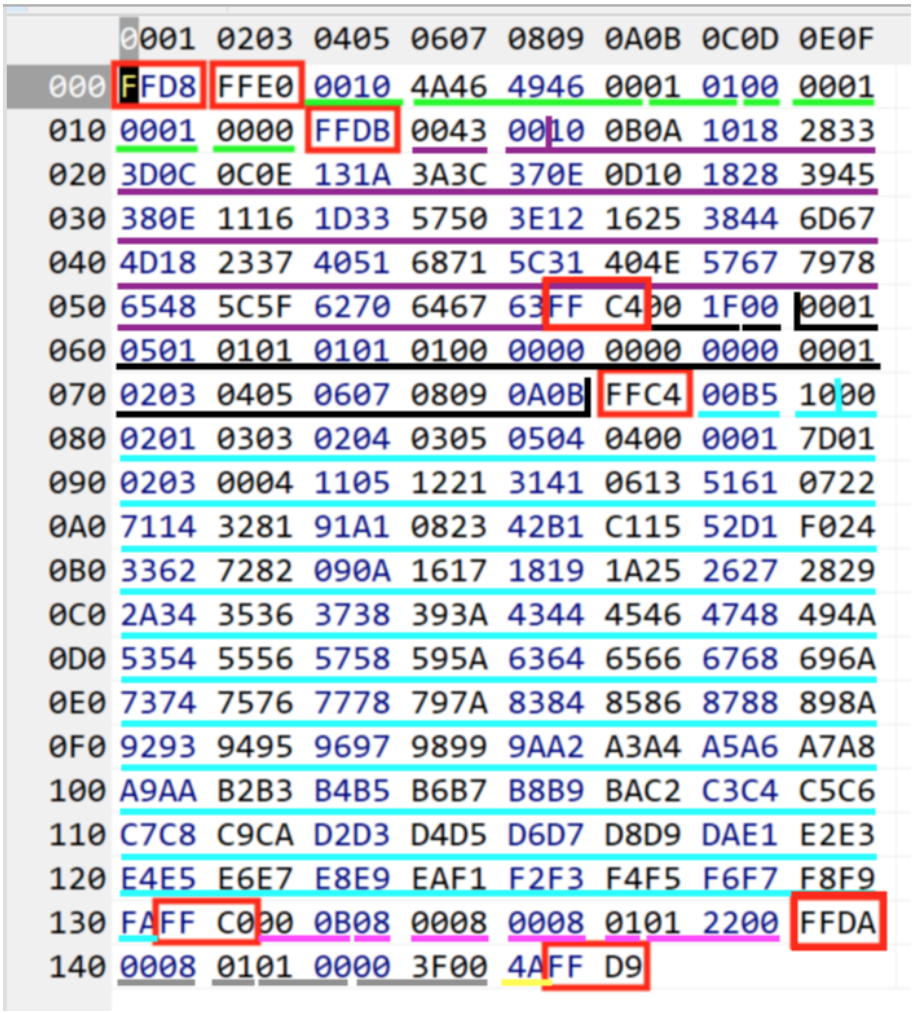
Hlavička souboru končí značkou EOI (End of Image), která se značí 0xD9.

Tabulka 10: Značka EOI v hlavičce souboru.

Název	Popis
Identifikátor značky	Identifikace EOI: 0xFF, 0xD9

```
mput(hex2dec('FFD9'),'usb');// identifikator znacky
mclose(soubor); //zavreme soubor
```

Výpis 19: Implementace značky EOI (End of Image).



Obrázek 24: Ukázka zapsaného souboru zobrazeného v programu PsPad.

Na výše uvedeném obrázku 24 můžeme vidět jednotlivé segmenty hlavičky souboru ve formátu JPEG. Červené čtverečky znázorňují jednotlivé značky a podtržítka povinné byty. Hlavička začíná značkou SOI, která se značí FFD8 a představuje začátek souboru.

Poté následuje značka APP0, která zobrazuje JFIF segment. Tu můžeme vidět pod označením FFE0. Za ní následují povinné byty (podtrženě zelenou barvou). První je délka, která je na 2 byty. V našem případě se jedná o délku 16 (v hexadecimální soustavě 10). Toto číslo se zapíše jako 0010. Následuje identifikátor souboru, který je na 5 bytů. Tyto byty se zapíšou jako 4A46494600. Dále následuje JFIF verze, která je na 2 byty. Tato verze se zapíše jako 0101. Následuje povinný byte jednotky hustoty x a y, v našem případě se jedná o 0 (specifikují se poměry stran). Ta se zapíše jako 00. Poté následují samotné hustoty x a y. Jelikož chceme rovnoměrné měřítko, obě hustoty budou nabývat hodnoty 1. Každá hustota se zapisuje na 2 byty, hustoty x a y budou zapsány jako 0001. Dále následují data náhledu, které jsou obvykle ignorovány, proto šířka i výška náhledu je rovna 0.

Následuje značka DQT, která zobrazuje zápis kvantizační tabulky a je označená pod FFDB. Za označením následují povinné byty (podtrženě fialovou barvou) délky značky. Délka značky je 67 (v hexadecimální soustavě 43), která se zapíše na 2 byty jako 0043. Dále následují QT informace, které jsou zapsány na 1 byte. Tyto informace se skládají z označení tabulky (0) a přesnosti na 8 bitů (0). Zapíše se tedy jako 00. Dále následuje samotná kvantizační tabulka.

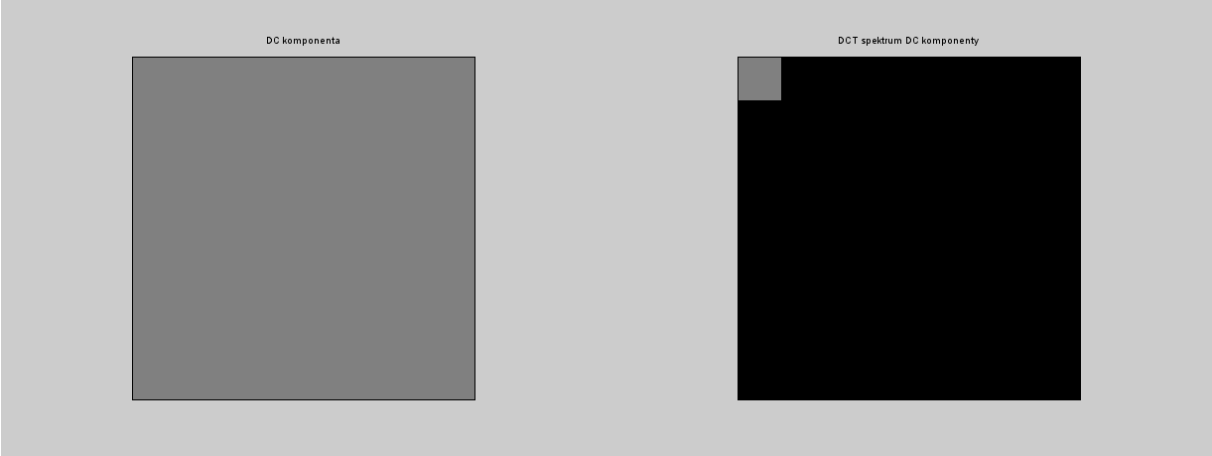
Dalším segmentem je DHT, která zapisuje jednotlivé tabulky pro Huffmanovo kódování. Značí se FFC4. V tomto případě zapisujeme pouze jasovou složku. Jsou zde tedy 2 tabulky. Jedna je pro stejnosměrnou složku a druhá je pro střídavou složku. První označení FFC4 je tabulka pro stejnosměrnou složku. Za tímto označením následují povinné byty (podtrženě černou barvou). Nejdříve délka segmentu, která specifikuje velikost tabulky pro Huffmanovo kódování. Ta je v našem případě 31 (v hexadecimální soustavě 1F). Musí se zapsat na 2 byty, proto následuje 001F. Dále následuje HT informace, které je zapsána na 1 byte. Tato informace se skládá z označení tabulky (0) a přesnosti na 8 bitů (0). Zapíše se tedy jako 00. Dále následuje samotná tabulka pro Huffmanovo kódování (pro stejnosměrnou složku). Dále následuje znovu označení FFC4 (podtrženě modrou barvou), které zobrazuje další tabulku pro Huffmanovo kódování (pro střídavé složky). Postup je stejný, délka segmentu je 181. Segment musí být zapsaný na 2 byty, tedy jako 00B5. Dále následuje HT informace, která je zapsána na 1 byte. Tato informace se skládá z označení tabulky (1) a přesnosti na 8 bitů (0). Zapíše se tedy jako 10. Dále následuje samotná tabulka pro Huffmanovo kódování (pro střídavé složky).

Po zápisu tabulek následuje segment SOF0, který se značí znaky FFC0 (podtrženě růžovou barvou). Jedná se o rámec, který udává různé parametry, např. výšku nebo šířku souboru. Následuje délka značky zapsaná na 2 byty, v našem případě je délka 11 (v hexadecimální soustavě 0B), která je zapsaná jako 000B. Následuje přesnost dat, k níž je potřeba 1 byte. Obvykle se zapisuje jako 08. Dalšími povinnými byty jsou výška a šířka obrázku, v našem případě se jedná o obrázek o velikosti 8×8. Šířka i výška je zapsaná na 2 byty, proto obě informace jsou zapsány jako 0008. Následuje počet komponent (velikost je 1 byte), v našem případě se jedná o 1 komponentu a to o jasovou složku, proto je zapsána jako 01. Dále jsou jednotlivé komponenty, které jsou na 3 byty. První byte označuje ID komponenty (jasová složka je 01), druhý byte označuje vzorkovací faktory 22, třetí byte je číslo kvantizační tabulky 00.

Předposledním segmentem je SOS, který zapisuje parametry týkající se kontroly, např. počet komponent. SOS značka je uvedena pod FFDA (podtrženě šedou barvou). Po identifikátoru opět následuje délka segmentu, která je na 2 byty. Tato délka se musí rovnat $6 + 2 \cdot$ počet komponent, v našem případě se rovná 8, zapíšeme jako 0008. Dále následuje počet komponent (1 byte), které využíváme (1, jasová složka). Zapíšeme jako 01. Dále následují 2 byty (ID = 1 jasová složka, ID = 0 tabulka pro Huffmanovo kódování), zapíšeme jako 0100. Poslední součástí značky jsou ignorovatelné byty, které mají velikost 3 byty. Zapíší se jako 003F00.

Po SOS segmentu následují komprimovaná data (podtrženě žlutou barvou). A poslední značkou, která určuje konec souboru EOI, je FFD9.

Uvedeme si několik příkladů na modelových datech. Na těchto datech otestujeme vytvořený kodér. Prvním příkladem bude zakódování a následné uložení souboru do formátu JPEG jedné stejnosměrné složky, kterou můžeme vidět na obrázku 25. Výsledky po kvantování DCT koeficientů můžeme vidět v tabulce 11.



Obrázek 25: DC komponenta a její DCT spektrum.

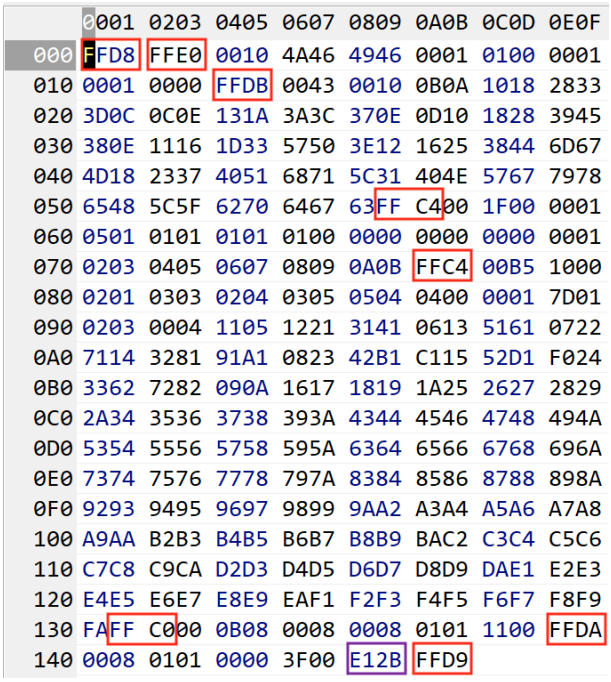
Tabulka 11: Výsledky po kvantizaci DCT koeficientů jedné stejnosměrné složky.

-59	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

$$\text{Intermediální sekvence} = (6, -59), (0, 0)$$

Stejnosměrná složka se skládá ze Symbol-1 a Symbol-2. Symbol-1 udává počet bitů k reprezentaci aktuální hodnoty koeficientu. Symbol-2 udává aktuální hodnotu koeficientu. Pokud při průchodu bloku nenarazíme na další nenulové číslo, ocitneme se na konci bloku. Konec bloku se značí dvojicí $(0, 0)$. Zakódování probíhá u každého symbolu odlišně. Symbol-1 se kóduje pomocí tabulky pro Huffmanovo kódování a Symbol-2 se kóduje jednotkovým doplňkem.

$$\text{Výsledný kód} = 1110\ 000100\ 1010$$

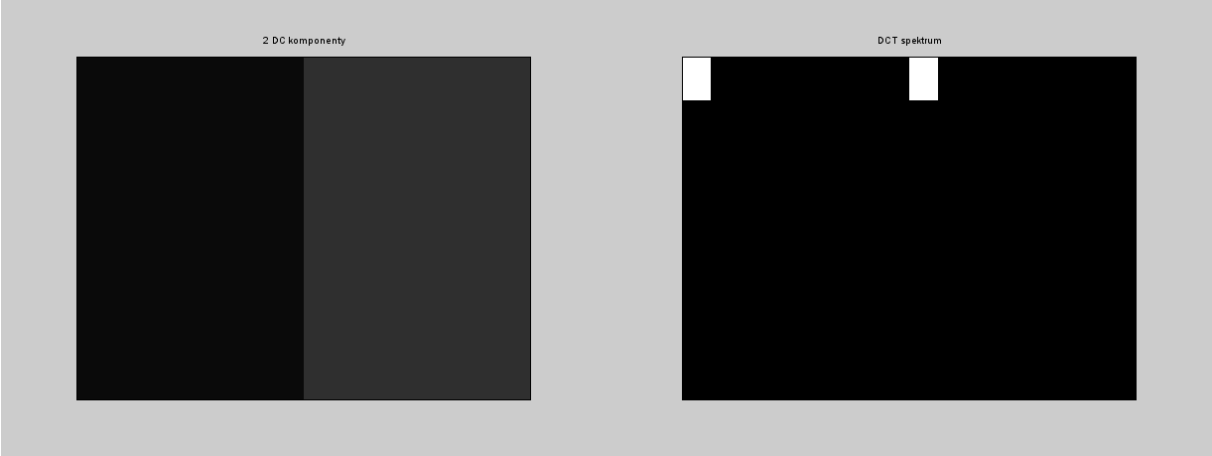


```
    0001 0203 0405 0607 0809 0A0B 0C0D 0E0F
000 FFD8 FFE0 0010 4A46 4946 0001 0100 0001
010 0001 0000 FFDB 0043 0010 0B0A 1018 2833
020 3D0C 0C0E 131A 3A3C 370E 0D10 1828 3945
030 380E 1116 1D33 5750 3E12 1625 3844 6D67
040 4D18 2337 4051 6871 5C31 404E 5767 7978
050 6548 5C5F 6270 6467 63FF C400 1F00 0001
060 0501 0101 0101 0100 0000 0000 0000 0001
070 0203 0405 0607 0809 0A0B FFC4 00B5 1000
080 0201 0303 0204 0305 0504 0400 0001 7D01
090 0203 0004 1105 1221 3141 0613 5161 0722
0A0 7114 3281 91A1 0823 42B1 C115 52D1 F024
0B0 3362 7282 090A 1617 1819 1A25 2627 2829
0C0 2A34 3536 3738 393A 4344 4546 4748 494A
0D0 5354 5556 5758 595A 6364 6566 6768 696A
0E0 7374 7576 7778 797A 8384 8586 8788 898A
0F0 9293 9495 9697 9899 9AA2 A3A4 A5A6 A7A8
100 A9AA B2B3 B4B5 B6B7 B8B9 BAC2 C3C4 C5C6
110 C7C8 C9CA D2D3 D4D5 D6D7 D8D9 DAE1 E2E3
120 E4E5 E6E7 E8E9 EAF1 F2F3 F4F5 F6F7 F8F9
130 FAFF C000 0B08 0008 0008 0101 1100 FFDA
140 0008 0101 0000 3F00 E12B FFD9
```

Obrázek 26: Ukázka zapsaného souboru jedné stejnosměrné složky zobrazeného v programu PsPad.

Na výše uvedeném obrázku 26 můžeme vidět zapsaný soubor. Červeně označené byty ukazují jednotlivé značky hlavičky a fialově zdůrazněné byty zobrazují samotná komprimovaná data. Výsledný kód je nutné rozdělit po osmi bitech. Pokud je délka poslední posloupnosti bitů menší než osm, doplní se jedničkami. Poté jej převedeme do hexadecimální soustavy.

Druhým příkladem bude zakódování a následné uložení souboru do formátu JPEG dvou stejnosměrných složek, které můžeme vidět na obrázku 27. Výsledky po kvantování DCT koeficientů dvou stejnosměrných složek můžeme vidět v tabulce 12.



Obrázek 27: Dvě DC komponenty (vlevo) a jejich DCT spektrum (vpravo).

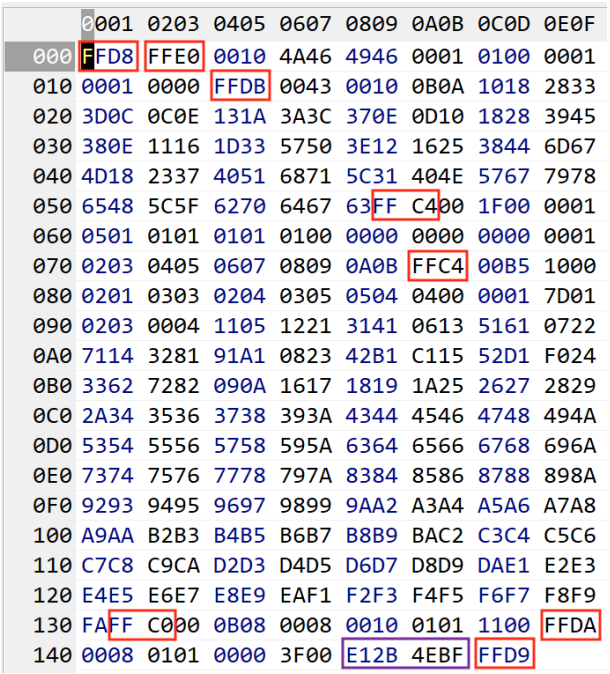
Tabulka 12: Výsledky po kvantizaci DCT koeficientů dvou stejnosměrných složek.

-59	0	0	0	0	0	0	0	-40	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Intermediální sekvence se sestavuje stejným způsobem, jako v předchozím případě. Začátek sekvence bude vypadat $(6, -59), (0, 0)$, dále přejdeme ke druhému bloku, kde máme druhou stejnosměrnou složku. V tomhle případě se složka vypočítá pomocí rovnice (21) $-40 - (-59) = 19$.

$$\text{Intermediální sekvence} = (6, -59), (0, 0), (5, 19), (0, 0)$$

$$\text{Výsledný kód} = 1110\ 000100\ 1010\ 110\ 10011\ 1010$$

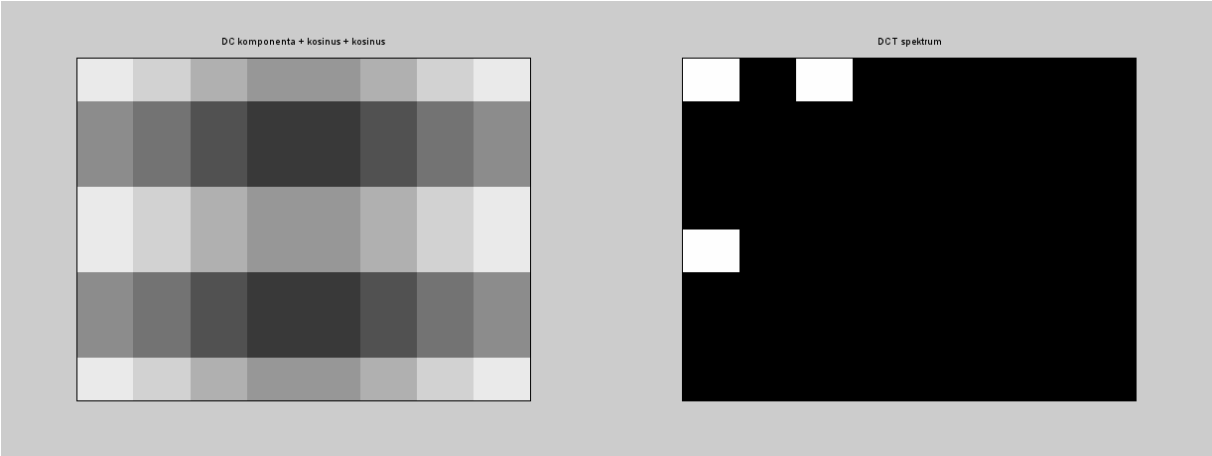


```
     0001 0203 0405 0607 0809 0A0B 0C0D 0E0F
000 FFD8 FFE0 0010 4A46 4946 0001 0100 0001
010 0001 0000 FFDB 0043 0010 0B0A 1018 2833
020 3D0C 0C0E 131A 3A3C 370E 0D10 1828 3945
030 380E 1116 1D33 5750 3E12 1625 3844 6D67
040 4D18 2337 4051 6871 5C31 404E 5767 7978
050 6548 5C5F 6270 6467 63FF C400 1F00 0001
060 0501 0101 0101 0100 0000 0000 0000 0001
070 0203 0405 0607 0809 0A0B FFC4 00B5 1000
080 0201 0303 0204 0305 0504 0400 0001 7D01
090 0203 0004 1105 1221 3141 0613 5161 0722
0A0 7114 3281 91A1 0823 42B1 C115 52D1 F024
0B0 3362 7282 090A 1617 1819 1A25 2627 2829
0C0 2A34 3536 3738 393A 4344 4546 4748 494A
0D0 5354 5556 5758 595A 6364 6566 6768 696A
0E0 7374 7576 7778 797A 8384 8586 8788 898A
0F0 9293 9495 9697 9899 9AA2 A3A4 A5A6 A7A8
100 A9AA B2B3 B4B5 B6B7 B8B9 BAC2 C3C4 C5C6
110 C7C8 C9CA D2D3 D4D5 D6D7 D8D9 DAE1 E2E3
120 E4E5 E6E7 E8E9 EAF1 F2F3 F4F5 F6F7 F8F9
130 FAFF C000 0B08 0008 0010 0101 1100 FFDA
140 0008 0101 0000 3F00 E12B 4EBF FFD9
```

Obrázek 28: Ukázka zapsaného souboru dvou stejnosměrných složek zobrazeného v programu PsPad.

Na výše zobrazeném obrázku 28 můžeme opět vidět zapsaný soubor, který je o dva byty větší. Je to tím, že výsledný kód je delší (intermediální sekvence obsahuje dvě DC složky). Červeně označené byty ukazují jednotlivé značky hlavičky a fialově zdůrazněné byty zobrazují samotná komprimovaná data.

Posledním příkladem bude sestavení z jedné stejnosměrné složky a dvou střídavých složek intermediální sekvence, její zakódování a následné uložení.



Obrázek 29: DC komponenta a součet dvou kosinových vln.

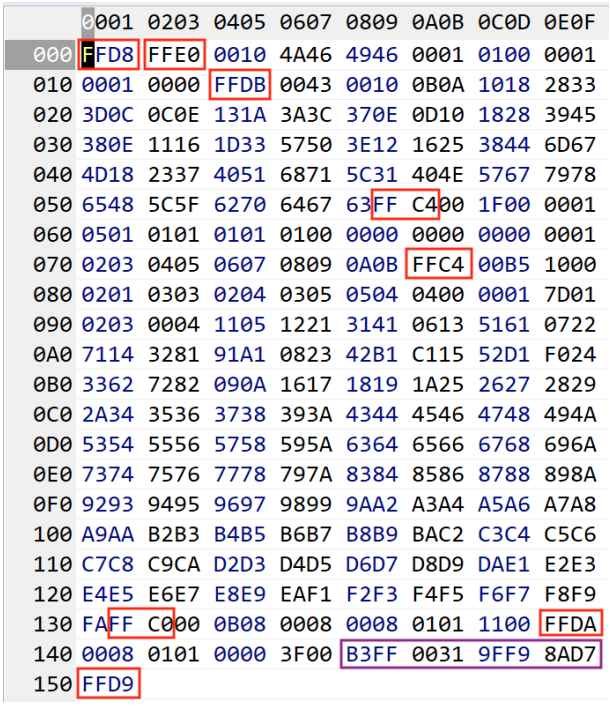
Tabulka 13: Výsledky po kvantizaci DCT koeficientů jedné stejnosměrné složky a dvou střídavých složek.

9	0	25	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

$$\text{Intermediální sekvence} = (4,9),(4,5)(25),(4,5)(21),(0,0)$$

$$\text{Výsledný kód} = 101\ 1001\ 1111111110011000\ 11001\ 1111111110011000\ 10101\ 1010$$

U střídavé složky Symbol-1 obsahuje dvě informace. První obsahuje informaci o počtu nul, které se nachází před nenulovou hodnotou. Druhá obsahuje opět počet bitů, které jsou potřeba k reprezentaci aktuální hodnoty koeficientu. Symbol-2 obsahuje aktuální hodnotu koeficientu. Jak již bylo řečeno, Symbol-1 se kóduje pomocí tabulek pro Huffmanovo kódování (i v případě střídavých složek). V tabulce 14 si vyhledáme dvojici $(4,5)$ (Symbol-1) a nalezneme kódové slovo 1111111110011000.



Obrázek 30: Ukázka zapsaného souboru jedné stejnosměrné složky a dvou střídavých složek zobrazeného v programu PsPad.

Na obrázku 30 můžeme opět vidět hlavičku souboru, která se až na komprimovaná data od předchozích příkladů neliší. S postupným narůstáním koeficientů můžeme vidět nárůst velikosti souboru (užitečná data se zvětšují).

Nyní se podíváme na obrázek 29 uložený ve formátu JPEG v porovnání se soubory uloženými pomocí funkce `imwrite` ve Scilabu, Matlabu a Octave. Na následujícím obrázku můžeme vidět soubory zobrazené v programu PsPad.

Obrázek 31: Ukázka jednotlivých souborů v programu PsPad.

Na výše uvedeném obrázku 31 můžeme vidět zapsání souboru ve čtyřech provedeních (pouze jasová složka). Nejdříve si srovnáme Můj kodér a funkci `imwrite` v Matlabu. V tomhle případě byla použita kvantizační tabulka, která je využívána ve funkci `imwrite` v Matlabu. Důvodem je, co nejvíce se přiblížit stejným výsledkům při srovnání. Pokud by se použila kvantizační tabulka, kterou standard doporučuje, lišila by se nejen v hlavičce kvantizační tabulka, ale také v samotných datech, jako je tomu v případě funkce `imwrite` ve Scilabu. Také se kvůli podobnosti souboru vyměnilo pořadí značek. Jak již bylo uvedeno dříve, DQT, DHT a SOF se mohou řadit v libovolném pořadí, ale budou zaznamenány po APP0 a před SOS. Ve funkci `imwrite` v Matlabu je pořadí značek DQT, SOF, a poté DHT.

Při srovnání prvních dvou souborů můžeme vidět, že jsou soubory stejné do značky FFDA, kde se liší komprimovaná data. Po bytech 3F00 začínají komprimovaná data. První 4 byty jsou stejné a poté se 4 byty liší i přes to, že je použita stejná kvantizační tabulka a tabulky pro Huffmanovo kódování.

Oproti tomu soubor, který je vytvořený pomocí funkce `imwrite` ve Scilabu, se liší i ve značce FFDB, kde můžeme vidět jinou kvantizační tabulku. To způsobuje, že se samotná data liší. Dále máme soubor uložený pomocí `imwrite` v Octave. Zde můžeme vidět velký rozdíl ve velikosti souboru. Funkce `imwrite` v Octave používá stejnou kvantizační tabulku ale jiné tabulky pro Huffmanovo kódování, jsou výrazně menší.

Pro získání více informací o souboru byla vyzkoušena funkce `imfinfo` v programu Matlab a Octave.

info

1x1 struct with 14 fields

Field	Value
Filename	'/Users/tynunka/Desktop/1DC2AC_muj_koder.jpg'
FileModDate	'20-Apr-2018 17:18:18'
FileSize	338
Format	'jpg'
FormatVersion	''
Width	8
Height	8
BitDepth	8
ColorType	'grayscale'
FormatSignature	''
NumberOfSamples	1
CodingMethod	'Huffman'
CodingProcess	'Sequential'
Comment	*0x0 cell*

Obrázek 32: Ukázka informací o souboru uložených pomocí mého kodéru, zobrazených pomocí funkce `imfinfo` v Matlabu.

info

1x1 struct with 14 fields

Field	Value
Filename	'/Users/tynunka/Desktop/1D2AC_matlab.jpg'
FileModDate	'20-Apr-2018 17:23:12'
FileSize	338
Format	'jpg'
FormatVersion	''
Width	8
Height	8
BitDepth	8
ColorType	'grayscale'
FormatSignature	''
NumberOfSamples	1
CodingMethod	'Huffman'
CodingProcess	'Sequential'
Comment	*0x0 cell*

Obrázek 33: Ukázka informací o souboru uložených pomocí funkce `imwrite` v Matlabu, zobrazených pomocí funkce `imfinfo` v Matlabu.

info

1x1 struct with 14 fields

Field	Value
Filename	'/Users/tynunka/Desktop/1D2AC_scilab.jpg'
FileModDate	'20-Apr-2018 17:18:18'
FileSize	338
Format	'jpg'
FormatVersion	''
Width	8
Height	8
BitDepth	8
ColorType	'grayscale'
FormatSignature	''
NumberOfSamples	1
CodingMethod	'Huffman'
CodingProcess	'Sequential'
Comment	*0x0 cell*

Obrázek 34: Ukázka informací o souboru uložených pomocí funkce `imwrite` ve Scilabu, zobrazených pomocí funkce `imfinfo` v Matlabu.

info

1x1 struct with 14 fields

Field	Value
Filename	'/Users/tynunka/Desktop/1D2AC_octave.jpg'
FileModDate	'20-Apr-2018 17:26:52'
FileSize	162
Format	'jpg'
FormatVersion	''
Width	8
Height	8
BitDepth	8
ColorType	'grayscale'
FormatSignature	''
NumberOfSamples	1
CodingMethod	'Huffman'
CodingProcess	'Sequential'
Comment	*0x0 cell*

Obrázek 35: Ukázka informací o souboru uložených pomocí funkce `imwrite` v Octave, zobrazených pomocí funkce `imfinfo` v Matlabu.

Na výše uvedených obrázcích jsou zobrazeny informace o jednotlivých souborech. Můžeme vidět informace o velikosti obrázku, použitých kódovacích metodách, typu barvy, ve kterém je obrázek uložen a formátu obrázku. Jak již bylo řečeno, soubor, který používá funkci `imwrite` v Octave, je menší o 176 bytů.

Další informace o souboru jsou ukázány na následujícím Obrázku 36 při použití funkce `imfinfo` v programu Octave. Jsou zde zobrazeny informace o souboru Můj kodér.

```
info =

  scalar structure containing the fields:

    Filename = C:\Users\tynunka\Documents\1DC2AC_muj_koder.jpg
    FileModDate = 20-Apr-2018 17:18:18
    FileSize =  338
    Format = JPEG
    FormatVersion =
    Width =  8
    Height =  8
    BitDepth =  8
    ColorType = grayscale
    DelayTime = 0
    DisposalMethod =
    LoopCount = 0
    ByteOrder = undefined
    Gamma = 0
    Chromaticities = [](1x0)
    Comment =
    Quality =  75
    Compression = undefined
    Colormap = [](0x0)
    Orientation =  1
    ResolutionUnit = undefined
    XResolution = 0
    YResolution = 0
    Software =
    Make =
    Model =
    DateTime =
    ImageDescription =
    Artist =
    Copyright =
    DigitalCamera =
```

Obrázek 36: Ukázka informací o souboru uložených pomocí mého kodéru, zobrazených pomocí funkce `imfinfo` v Octave.

Zde můžeme vidět, že se jedná o černobílý formát JPEG, velikost souboru je 338 bytů a rozlišení souboru je 8×8. Další soubory jsou uvedeny v příloze D.

I přes důkladné zkoumání metadat souborů zjištěných pomocí výše popsané funkce (v programech Matlab a Octave) nebyly zjištěny žádné informace, které by napovídaly tomu, proč se komprimovaná data souborů liší. A to i přes použití stejné kvantizační tabulky a tabulky pro Huffmanovo kódování.

Závěr

Cílem této diplomové práce bylo provést rešerši nástrojů pro zpracování obrazu v opensourceovém prostředí Scilab. Poté pomocí jednoho konkrétního nástroje (po dohodě s vedoucím práce) detailně popsat kodér formátu JPEG pro účely výuky.

V první části se práce věnuje rešerši dostupných nástrojů pro zpracování obrazu. Při výběru nástroje byl kladen důraz na aktivitu vývoje, četnost aktualizací, počet stažení a míry podpory s poslední verzí Scilabu. Po dohodě s vedoucím práce byl vybrán nástroj IPCV, který je kompatibilní s poslední verzí Scilabu (verze 6.0.0), a také má obsáhlejší dokumentaci, která byla nápomocná při řešení problémů.

Dále je uveden detailní popis jednotlivých kroků komprese JPEG. V každé podkapitole se plně věnuji dané problematice, přičemž jsou uvedeny příklady s podrobným popisem formou komentářů uvnitř implementace.

Při zpracování mé diplomové práce byl největší problém v kapitole Ukládání. Při testování funkčnosti jsem narazila na problém, kdy se komprimovaná data lišila i přesto, že byl zpracovávaný obraz ukládán dle standardu JPEG. Pro porovnávání souborů jsem použila funkci `imwrite` v Matlabu, Scilabu a Octave. Abych dosáhla co největší podobnosti se souborem vytvořeným pomocí funkce `imwrite`, aplikovala jsem stejnou kvantizační tabulku a změnila pořadí značek DQT, DHT a SOF přesně tak, jak je uvedeno ve výše zmíněné funkci. Poté se hlavička souboru shodovala i s prvními čtyřmi byty komprimovaných dat, zbytek dat byl jiný. Nakonec jsem ověřila pomocí funkce `imfinfo` v Matlabu a Octave, že se metadata testovacích souborů shodují. To, proč se daná komprimovaná data liší mezi mým programem a výstupy z výše uvedených, se bohužel nepodařilo ani po důkladném testování odhalit.

Literatura

[1] ONDRUŠ, Jan. Bezztrátová komprese JPEG grafiky [online]. Praha, 2009 [cit.2018-03-02]. Dostupné z: https://is.cuni.cz/webapps/zzp/detail/49840. DIPLOMOVÁ PRÁCE. Univerzita Karlova v Praze, Matematicko-fyzikální fakulta. Vedoucí práce Mgr. Jan Lánský.

[2] BRIGGS, William L. a Van Emden HENSON. The DFT: an owner's manual for the discrete Fourier transform. Philadelphia: Society for Industrial and Applied Mathematics, c1995. ISBN 0-89871-342-0.

[3] HORÁK, David. DISKRÉTNÍ TRANSFORMACE: Matematika pro inženýry 21. století [online]. 2012 [cit. 2018-03-10]. Vysoká škola báňská – Technická univerzita Ostrava a Západočeská univerzita v Plzni.

[4] WALACH, Jan. Aplikace Fourierovy analýzy na rozpoznávání kvality samohlásek podle jejich formantů [online]. Olomouc, 2013 [cit. 2018-03-17]. Dostupné z: https://theses.cz/id/gxrdcg/00176001-637083732.pdf. BAKALÁŘSKÁ PRÁCE. UNIVERZITA PALACKÉHO V OLOMOUCI, PŘÍRODOVĚDECKÁ FAKULTA. Vedoucí práce RNDr. Tomáš Fürst, Ph.D.

[5] KRISHNA A.N., Radha a G.SHRUTHI. JPEG Encoder using Discrete Cosine Transform & Inverse Discrete Cosine Transform [online]. 1.4.2013 [cit. 2018-03-19]. ISSN 2278-8735. Dostupné z: https://pdfs.semanticscholar.org/1e9d/4ee6b5cbf442bd9e4a4c3cf4d1586045a8ce.pdf

[6] TECHNICAL STANDARDIZATION COMMITTEE ON AV & IT STORAGE SYSTEMS AND EQUIPMENT. Exchangeable image file format for digital still cameras: Exif Version 2.2 [online]. Japan Electronics and Information Technology Industries Association, 2002 [cit. 2018-03-22]. Dostupné z: http://www.exif.org/Exif2-2.PDF

[7] JPEG File Layout and Format: The File Layout [online]. [cit. 2018-03-28]. Dostupné z: http://vip.sugovica.hu/Sardi/kepnezo/JPEG%20File%20Layout%20and%20Format.htm

[8] TIŠNOVSKÝ, Pavel. Ztrátová komprese obrazových dat pomocí JPEG [online]. 14. 12. 2006 [cit. 2018-04-01]. Dostupné z: https://www.root.cz/clanky/ztratova-komprese-obrazovych-dat-pomoci-jpeg/

[9] TIŠNOVSKÝ, Pavel. Programujeme JPEG: transformace a podvzorkování barev [online]. 21. 12. 2006 [cit. 2018-04-03]. Dostupné z: https://www.root.cz/clanky/programujeme-jpeg-transformace-a-podvzorkovani-barev/

[10] TIŠNOVSKÝ, Pavel. Programujeme JPEG: diskrétní kosinová transformace (DCT) [online]. 4. 1. 2007 [cit. 2018-04-03]. Dostupné z: https://www.root.cz/clanky/programujeme-jpeg-diskretni-kosinova-transformace-dct/

[11] TIŠNOVSKÝ, Pavel. Programujeme JPEG: Kvantizace DCT koeficientů [online]. 11. 1. 2007 [cit. 2018-04-03]. Dostupné z: https://www.root.cz/clanky/programujeme-jpeg-kvantizace-dct-koeficientu/

[12] TIŠNOVSKÝ, Pavel. Programujeme JPEG: Huffmanovo kódování kvantovaných DCT složek [online]. 18.1.2007 [cit. 2018-04-03]. Dostupné z: https://www.root.cz/clanky/programujeme-jpeg-huffmanovo-kodovani-kvantovanych-dct-slozek/

[13] TIŠNOVSKÝ, Pavel. Programujeme JPEG: Interní struktura souborů typu JFIF/JPEG [online]. 25. 1. 2007 [cit. 2018-04-03]. Dostupné z: https://www.root.cz/clanky/programujeme-jpeg-interni-struktura-souboru-typu-jfifjpeg/

[14] TIŠNOVSKÝ, Pavel. Programujeme JPEG: Načtení informací ze souborů typu JFIF/JPEG [online]. 1. 2. 2007 [cit. 2018-04-03]. Dostupné z: https://www.root.cz/clanky/programujeme-jpeg-nacteni-informaci-ze-souboru-typu-jfifjpeg/

[15] SKAPA, Jan. Zpracování číslicových signálů: Pracovní verze [online]. Ostrava [cit. 2018-04-08]. Dostupné z: https://owncloud.cesnet.cz/index.php/s/542ac2fa24c0ad19f011585ccce01bc5?path=%2F0000_Literatura#pdfviewer. Skripta. Vysoká škola báňská – Technická univerzita Ostrava.

[16] NEVŘIVA, Pavel a Martin PIEŠ. SIGNÁLY A SOUSTAVY [online]. Ostrava, 2012 [cit. 2018-04-10]. Dostupné z: https://lms.vsb.cz/pluginfile.php/260756/mod_resource/content/1/ESF_SAS_skripta_V9.pdf. Učební text a návody do cvičení. Vysoká škola báňská - Technická univerzita Ostrava.

[17] TERMINAL EQUIPMENT AND PROTOCOLS FOR TELEMATIC SERVICES: INFORMATION TECHNOLOGY – DIGITAL COMPRESSION AND CODING OF CONTINUOUS-TONE STILL IMAGES – REQUIREMENTS AND GUIDELINES [online]. [cit. 2018-04-10]. Dostupné z: https://www.w3.org/Graphics/JPEG/itu-t81.pdf

A Tabulky pro Huffmanovo kódování (jasovou složku)

Tabulka 14: Tabulka pro koeficienty AC (jas).

Počet nul/Velikost hodnoty v bitech	Délka kódu	Kódové slovo
0/0	4	1010
0/1	2	00
0/2	2	01
0/3	3	100
0/4	4	1011
0/5	5	11010
0/6	7	1111000
0/7	8	11111000
0/8	10	1111110110
0/9	16	1111111110000010
0/A	16	1111111110000011
1/1	4	1100
1/2	5	11011
1/3	7	1111001
1/4	9	111110110
1/5	11	11111110110
1/6	16	1111111110000100
1/7	16	1111111110000101
1/8	16	1111111110000110
1/9	16	1111111110000111
1/A	16	1111111110001000
2/1	5	11100
2/2	8	11111001
2/3	10	1111110111
2/4	12	111111110100

2/5	16	1111111110001001
2/6	16	1111111110001010
2/7	16	1111111110001011
2/8	16	1111111110001100
2/9	16	1111111110001101
2/A	16	1111111110001110
3/1	6	111010
3/2	9	111110111
3/3	12	111111110101
3/4	16	1111111110001111
3/5	16	1111111110010000
3/6	16	1111111110010001
3/7	16	1111111110010010
3/8	16	1111111110010011
3/9	16	1111111110010100
3/A	16	1111111110010101
4/1	6	111011
4/2	10	1111111000
4/3	16	1111111110010110
4/4	16	1111111110010111
4/5	16	1111111110011000
4/6	16	1111111110011001
4/7	16	1111111110011010
4/8	16	1111111110011011
4/9	16	1111111110011100
4/A	16	1111111110011101
5/1	7	1111010
5/2	11	11111110111

5/3	16	1111111110011110
5/4	16	1111111110011111
5/5	16	1111111110100000
5/6	16	1111111110100001
5/7	16	1111111110100010
5/8	16	1111111110100011
5/9	16	1111111110100100
5/A	16	1111111110100101
6/1	7	1111011
6/2	12	111111110110
6/3	16	1111111110100110
6/4	16	1111111110100111
6/5	16	1111111110101000
6/6	16	1111111110101001
6/7	16	1111111110101010
6/8	16	1111111110101011
6/9	16	1111111110101100
6/A	16	1111111110101101
7/1	8	11111010
7/2	12	111111110111
7/3	16	1111111110101110
7/4	16	1111111110101111
7/5	16	1111111110110000
7/6	16	1111111110110001
7/7	16	1111111110110010
7/8	16	1111111110110011
7/9	16	1111111110110100
7/A	16	1111111110110101

8/1	9	111111000
8/2	15	111111111000000
8/3	16	1111111110110110
8/4	16	1111111110110111
8/5	16	1111111110111000
8/6	16	1111111110111001
8/7	16	1111111110111010
8/8	16	1111111110111011
8/9	16	1111111110111100
8/A	16	1111111110111101
9/1	9	111111001
9/2	16	1111111110111110
9/3	16	1111111110111111
9/4	16	1111111111000000
9/5	16	1111111111000001
9/6	16	1111111111000010
9/7	16	1111111111000011
9/8	16	1111111111000100
9/9	16	1111111111000101
9/A	16	1111111111000110
A/1	9	111111010
A/2	16	1111111111000111
A/3	16	1111111111001000
A/4	16	1111111111001001
A/5	16	1111111111001010
A/6	16	1111111111001011
A/7	16	1111111111001100
A/8	16	1111111111001101

A/9	16	1111111111001110
A/A	16	1111111111001111
B/1	10	1111111001
B/2	16	1111111111010000
B/3	16	1111111111010001
B/4	16	1111111111010010
B/5	16	1111111111010011
B/6	16	1111111111010100
B/7	16	1111111111010101
B/8	16	1111111111010110
B/9	16	1111111111010111
B/A	16	1111111111011000
C/1	10	1111111010
C/2	16	1111111111011001
C/3	16	1111111111011010
C/4	16	1111111111011011
C/5	16	1111111111011100
C/6	16	1111111111011101
C/7	16	1111111111011110
C/8	16	1111111111011111
C/9	16	1111111111100000
C/A	16	1111111111100001
D/1	11	11111111000
D/2	16	1111111111100010
D/3	16	1111111111100011
D/4	16	1111111111100100
D/5	16	1111111111100101
D/6	16	1111111111100110

D/7	16	1111111111100111
D/8	16	1111111111101000
D/9	16	1111111111101001
D/A	16	1111111111101010
E/1	16	1111111111101011
E/2	16	1111111111101100
E/3	16	1111111111101101
E/4	16	1111111111101110
E/5	16	1111111111101111
E/6	16	1111111111110000
E/7	16	1111111111110001
E/8	16	1111111111110010
E/9	16	1111111111110011
E/A	16	1111111111110100
F/0	11	11111111001
F/1	16	1111111111110101
F/2	16	1111111111110110
F/3	16	1111111111110111
F/4	16	1111111111111000
F/5	16	1111111111111001
F/6	16	1111111111111010
F/7	16	1111111111111011
F/8	16	1111111111111100
F/9	16	1111111111111101
F/A	16	1111111111111110

B Tabulky pro Huffmanovo kódování (barvonosné složky)

Tabulka 15: Tabulka pro koeficienty AC (chrominance).

Počet nul/Velikost hodnoty v bitech	Délka kódu	Kódové slovo
0/0	2	00
0/1	2	01
0/2	3	100
0/3	4	1010
0/4	5	11000
0/5	5	11001
0/6	6	111000
0/7	7	1111000
0/8	9	111110100
0/9	10	1111110110
0/A	12	111111110100
1/1	4	1011
1/2	6	111001
1/3	8	11110110
1/4	9	111110101
1/5	11	11111110110
1/6	12	111111110101
1/7	16	1111111110001000
1/8	16	1111111110001001
1/9	16	1111111110001010
1/A	16	1111111110001011
2/1	5	11010
2/2	8	11110111
2/3	10	1111110111
2/4	12	111111110110

2/5	15	111111111000010
2/6	16	1111111110001100
2/7	16	1111111110001101
2/8	16	1111111110001110
2/9	16	1111111110001111
2/A	16	1111111110010000
3/1	5	11011
3/2	8	11111000
3/3	10	1111111000
3/4	12	111111110111
3/5	16	1111111110010001
3/6	16	1111111110010010
3/7	16	1111111110010011
3/8	16	1111111110010100
3/9	16	1111111110010101
3/A	16	1111111110010110
4/1	6	111010
4/2	9	111110110
4/3	16	1111111110010111
4/4	16	1111111110011000
4/5	16	1111111110011001
4/6	16	1111111110011010
4/7	16	1111111110011011
4/8	16	1111111110011100
4/9	16	1111111110011101
4/A	16	1111111110011110
5/1	6	111011
5/2	10	1111111001

5/3	16	1111111110011111
5/4	16	1111111110100000
5/5	16	1111111110100001
5/6	16	1111111110100010
5/7	16	1111111110100011
5/8	16	1111111110100100
5/9	16	1111111110100101
5/A	16	1111111110100110
6/1	7	1111001
6/2	11	11111110111
6/3	16	1111111110100111
6/4	16	1111111110101000
6/5	16	1111111110101001
6/6	16	1111111110101010
6/7	16	1111111110101011
6/8	16	1111111110101100
6/9	16	1111111110101101
6/A	16	1111111110101110
7/1	7	1111010
7/2	11	11111111000
7/3	16	1111111110101111
7/4	16	1111111110110000
7/5	16	1111111110110001
7/6	16	1111111110110010
7/7	16	1111111110110011
7/8	16	1111111110110100
7/9	16	1111111110110101
7/A	16	1111111110110110

8/1	8	11111001
8/2	16	1111111110110111
8/3	16	1111111110111000
8/4	16	1111111110111001
8/5	16	1111111110111010
8/6	16	1111111110111011
8/7	16	1111111110111100
8/8	16	1111111110111101
8/9	16	1111111110111110
8/A	16	1111111110111111
9/1	9	111110111
9/2	16	1111111111000000
9/3	16	1111111111000001
9/4	16	1111111111000010
9/5	16	1111111111000011
9/6	16	1111111111000100
9/7	16	1111111111000101
9/8	16	1111111111000110
9/9	16	1111111111000111
9/A	16	1111111111001000
A/1	9	111111000
A/2	16	1111111111001001
A/3	16	1111111111001010
A/4	16	1111111111001011
A/5	16	1111111111001100
A/6	16	1111111111001101
A/7	16	1111111111001110
A/8	16	1111111111001111

A/9	16	1111111111010000
A/A	16	1111111111010001
B/1	9	111111001
B/2	16	1111111111010010
B/3	16	1111111111010011
B/4	16	1111111111010100
B/5	16	1111111111010101
B/6	16	1111111111010110
B/7	16	1111111111010111
B/8	16	1111111111011000
B/9	16	1111111111011001
B/A	16	1111111111011010
C/1	9	111111010
C/2	16	1111111111011011
C/3	16	1111111111011100
C/4	16	1111111111011101
C/5	16	1111111111011110
C/6	16	1111111111011111
C/7	16	1111111111100000
C/8	16	1111111111100001
C/9	16	1111111111100010
C/A	16	1111111111100011
D/1	11	11111111001
D/2	16	1111111111100100
D/3	16	1111111111100101
D/4	16	1111111111100110
D/5	16	1111111111100111
D/6	16	1111111111101000

D/7	16	1111111111101001
D/8	16	1111111111101010
D/9	16	1111111111101011
D/A	16	1111111111101100
E/1	14	11111111100000
E/2	16	1111111111101101
E/3	16	1111111111101110
E/4	16	1111111111101111
E/5	16	1111111111110000
E/6	16	1111111111110001
E/7	16	1111111111110010
E/8	16	1111111111110011
E/9	16	1111111111110100
E/A	16	1111111111110101
F/0	10	1111111010
F/1	15	111111111000011
F/2	16	1111111111110110
F/3	16	1111111111110111
F/4	16	1111111111111000
F/5	16	1111111111111001
F/6	16	1111111111111010
F/7	16	1111111111111011
F/8	16	1111111111111100
F/9	16	1111111111111101
F/A	16	1111111111111110

C Implementace vykreslení RGB kostky v RGB a YCbCr prostoru

```
//RGB
t = 0:7; //pocet bodu v dimenzi
t = t * 36.42; //naplneni bodu hodnotami 0 az 255 s krokem 36,42
x = [0:511]; y = [0:511]; z = [0:511]; //alokace vektoru pro celkovy pocet bodu (8*8*8)
color = matrix(1:1536,512,3); //barvy pro kazdy bod
//definovani souradnic pro vsech 512 bodu kombinaci vyse uvedeneho kroku
for i=0:511
   index = modulo(i,8) + 1;
   x(i + 1) = t(index);
end

for j=0:63
   for i=1:8
      y((8 * j) + i) = x(modulo(j, 8) + 1);
   end
end

for j=0:7
   for i=1:64
      z(i + (j * 64)) = x(j + 1);
   end
end
//definovani vektoru barev, ktere se rovnaji souradnicim jednotlivych bodu
for j=1:512
   color(j, 1:3) = [x(j), y(j), z(j)] ./ 256;
end
//vykresleni grafu a popsani os
figure();
scatter3(x, y, z, 25, color, "fill");
title('RGB'); xlabel('R'); ylabel('G'); zlabel('B');

// YCbCr
x_y = [0:511]; y_y = [0:511]; z_y = [0:511];//alokace vektoru pro celkovy pocet bodu (8*8*8)
for i=1:512
   x_y(i) = 0.299 * x(i) + 0.587 * y(i) + 0.114 * z(i) - 128; //vypocet Y
   y_y(i) = - 0.1687 * x(i) - 0.3313 * y(i) + 0.5 * z(i); //vypocet Cb
   z_y(i) = 0.5 * x(i) - 0.4187 * y(i) - 0.0813 * z(i); //vypocet Cr
end
//vykresleni grafu a popsani os
figure();
scatter3(x_y, y_y, z_y, 25, color, "fill");
title('YCbCr'); xlabel('Y'); ylabel('Yb'); zlabel('Yr');
```

Výpis 20: Implementace vykreslení RGB kostky v RGB a YCbCr prostoru.

D Informace o uloženém souboru získané pomocí funkce `imwrite` ve Scilabu, Matlabu a Octave

```
info =

  scalar structure containing the fields:

    Filename = C:\Users\tynunka\Documents\1D2AC_matlab.jpg
    FileModDate = 20-Apr-2018 17:23:12
    FileSize =  338
    Format = JPEG
    FormatVersion =
    Width =  8
    Height =  8
    BitDepth =  8
    ColorType = grayscale
    DelayTime = 0
    DisposalMethod =
    LoopCount = 0
    ByteOrder = undefined
    Gamma = 0
    Chromaticities = [](1x0)
    Comment =
    Quality =  75
    Compression = undefined
    Colormap = [](0x0)
    Orientation =  1
    ResolutionUnit = undefined
    XResolution = 0
    YResolution = 0
    Software =
    Make =
    Model =
    DateTime =
    ImageDescription =
    Artist =
    Copyright =
    DigitalCamera =
```

Obrázek 37: Ukázka informací o souboru uložených pomocí funkce `imwrite` v Matlabu, zobrazených pomocí funkce `imfinfo` v Octave.

```
info =

  scalar structure containing the fields:

    Filename = C:\Users\tynunka\Documents\1D2AC_scilab.jpg
    FileModDate = 20-Apr-2018 17:18:18
    FileSize =  338
    Format = JPEG
    FormatVersion =
    Width =  8
    Height =  8
    BitDepth =  8
    ColorType = grayscale
    DelayTime = 0
    DisposalMethod =
    LoopCount = 0
    ByteOrder = undefined
    Gamma = 0
    Chromaticities = [](1x0)
    Comment =
    Quality =  75
    Compression = undefined
    Colormap = [](0x0)
    Orientation =  1
    ResolutionUnit = undefined
    XResolution = 0
    YResolution = 0
    Software =
    Make =
    Model =
    DateTime =
    ImageDescription =
    Artist =
    Copyright =
    DigitalCamera =
```

Obrázek 38: Ukázka informací o souboru uložených pomocí funkce `imwrite` ve Scilabu, zobrazených pomocí funkce `imfinfo` v Octave.

```
info =

  scalar structure containing the fields:

    Filename = C:\Users\tynunka\Documents\1D2AC_octave.jpg
    FileModDate = 20-Apr-2018 17:26:52
    FileSize =  162
    Format = JPEG
    FormatVersion =
    Width =  8
    Height =  8
    BitDepth =  8
    ColorType = grayscale
    DelayTime = 0
    DisposalMethod =
    LoopCount = 0
    ByteOrder = undefined
    Gamma = 0
    Chromaticities = [](1x0)
    Comment =
    Quality =  75
    Compression = undefined
    Colormap = [](0x0)
    Orientation =  1
    ResolutionUnit = Inch
    XResolution =  72
    YResolution =  72
    Software =
    Make =
    Model =
    DateTime =
    ImageDescription =
    Artist =
    Copyright =
    DigitalCamera =
```

Obrázek 39: Ukázka informací o souboru uložených pomocí funkce `imwrite` v Octave, zobrazených pomocí funkce `imfinfo` v Octave.

E Skripty a funkce

Obsah této přílohy je na přiloženém CD